TESIS DOCTORAL

MODELO DE DISEÑO ORIENTADO A MARCOS PARA SISTEMAS BASADOS EN EL CONOCIMIENTO

presentada en la

FACULTAD DE INFORMATICA

de la

UNIVERSIDAD POLITECNICA DE MADRID

para la obtención del

GRADO DE DOCTOR EN INFORMATICA

AUTORA: Asunción Gómez Pérez

Licenciada en Informática por la

Universidad Politécnica de Madrid

DIRECTORA: Natalia Juristo Juzgado

Madrid, diciembre 1993

A mis Padres, Sole y Manolo

RESUMEN

El objetivo de esta Tesis es crear un **Modelo de Diseño Orientado a Marcos** que, intermedio entre el *Mundo Externo* y el *Modelo Interno* del Mundo que supone el sistema implementado, disminuya la pérdida de conocimiento que se produce al formalizar la realidad en Bases de Conocimientos.

El modelo disminuye la pérdida de conocimiento al formalizar Bases de Conocimiento, acercando el formalismo de Marcos al Mundo Externo, porque:

1. Crea una base teórica que uniformiza el concepto de Marco en el plano de la Formalización, estableciendo un conjunto de restricciones sintácticas y semánticas que impedirán, al Ingeniero del Conocimiento (IC) cuando formaliza, definir elementos no permitidos o el uso indebido de ellos.

2. Se incrementa la expresividad del formalismo al asociar a cada una de las propiedades de un marco clase un parámetro adicional que simboliza la *representatividad* de la propiedad en el concepto. Este parámetro, y las técnicas de inferencia que trabajan con él, permitirán al IC introducir en el *Modelo Formalizado* conocimiento que antes no introducía al construir la base de conocimientos y que, sin embargo, sí existía en la realidad.

3. Se propone una técnica de equiparación que trabaja con el conocimiento incierto presente en el dominio. Esta técnica de equiparación, utiliza la representatividad de las propiedades en los marcos clase y el grado de certeza de las propiedades de las entidades para calcular el valor de equiparación y, así, determinar en qué medida los marcos clase seleccionados son consistentes con la descripción de la situación actual dada por una entidad.

4. Proporciona nuevas técnicas de inferencia basadas en la transferencia de propiedades y modifica las ya existentes. Las transferencias de propiedades realizadas sobre relaciones "ad hoc" definidas por el IC al construir el sistema, es una nueva técnica de inferencia independiente y complementaria a la transferencia de propiedades llamada tradicionalmente *Herencia* (cesión de

propiedades entre padres e hijos). A esta nueva técnica, se le ha llamado *Donación*, es decir, cesión de propiedades entre marcos sin parentesco.

Como aportación práctica, se ha construido un entorno de construcción de Sistemas Basados en el Conocimiento formalizados en Marcos, donde se han introducido todos los nuevos conceptos del Modelo Teórico de la Tesis. Se trata de una cierta anidación. Es decir, son marcos que permiten formalizar cualquier SBC en marcos. El entorno permitirá al IC formalizar bases de conocimientos automáticamente y éste podrá validar el conocimiento del dominio en la fase de formalización en lugar de tener que esperar a que la BC esté implementada.

Todo ello lleva a describir el **Modelo de Diseño Orientado a Marcos** como un puente que aproxima y comunica el *Mundo Externo* con el *Modelo Interno* asociado a la realidad e implementado en una computadora, disminuyendo así las diversas pérdidas de conocimiento que si bien no ocurren simultáneamente al construir Sistemas Basados en el Conocimiento, sí coexisten en él.

ABSTRACT

The goal of this thesis is to create a **Frame-Oriented Design Model** that, bridging the *Outside World* and the implemented system's *Internal Model* of the World, reduces the amount of knowledge lost when reality is formalized in Knowledge Bases (KB).

The model diminishes the loss of knowledge when formalizing a KB and brings the Frame-formalized Model closer to the Outside World because:

1. It creates a theory that standardizes the concept of *frame* at the formalization level to establish a set of syntactic and semantic constraints that will prevent the Knowledge Engineer (KE) from defining forbidden elements or their undue use in the formalization process.

2. The formalism's expressiveness is increased by associating an additional parameter to each of the properties of a class frame to symbolize the *representativeness* of the concept property. This parameter and the related inference techniques will allow the KE to enter knowledge into the *Formalized Model* that actually existed but that was not used previously when building the KB.

3. The proposed technique involves matching and works with uncertain knowledge present in the domain. This matching technique takes the representativeness of the properties in the class frame and the degree of certainty of the properties of the entities to calculate the matching value and thus determine to what extent the class frames selected are consistent with the description of the present situation given by an entity.

4. It offers new inference techniques based on property transfer and alters existing ones. Property transfer on ad hoc relations defined by the KE when building a system is a new inference technique independent of and complementary to property transfer traditionally termed *Inheritance* (transfer of properties between parents and children). This new technique

has been called *Donation* (transfer of properties between frames without relationships).

5. It improves control of the procedural knowledge defined in the frames by introducing OO concepts.

A frame-formalized KBS building environment has been constructed, incorporating all the new concepts of the theoretical model set out in the thesis. There is some embedding, that is, they are frames that provide for any KBS to be formalized in frames. The environment will enable the KE to formalize KB automatically, and he will be able to validate the domain knowledge in the formalization stage instead of having to wait until the KB has been implemented.

This is a description of the **Frame-oriented Design Model**, a bridge that brings closer and communicates the *Outside World* with the *Internal Model* associated to reality and implemented on a computer, thus reducing the different losses in knowledge that, though they do not occur simultaneosly when building a Knowledge-based System, coexist within it.

AGRADECIMIENTOS

Deseo expresar mi más sincero agradecimiento a *Doña* Natalia Juristo, directora de esta Tesis, por su amistad, apoyo y colaboración prestada en todo momento en la elaboración de este trabajo, tercero de unas serie que comenzó hace varios años, y de otros trabajos presentes, pasados y futuros. También a su S.O. por su infinita paciencia con las dos.

También quiero mostrar mi agrademiento a D. Juan Pazos por su concienzuda labor en la revisión de este trabajo y por los fabulosos dibujos que realizó. Especialmente, por el espléndido *Bombón* que pintó.

Mención especial merecen mis padres, Sole y Manolo, que con buen talante y ánimo han sufrido mi ausencia durante la realización de este trabajo. A ellos mi agradecimiento por su apoyo continuado, por el ánimo infundido y por la ayuda prestada en los momentos críticos.

También deseo mostrar mi agradecimiento a los geniales pintores de mi familia que se prestaron a dibujar un conjunto de conceptos. Especialmente, nombrar a mi abuela Soledad, que a sus ochenta y seis años es una artista, a Caro, Jose M., Ana y Chema. También a mi amigo *Moreno*, por haberlos reproducido.

Quiero recordar explícitamente a mi abuelo Julio, quien hubiera disfrutado enormemente en estos días.

Igualmente, a todas las personas de la Facultad de Informática que han contribuido a la realización de este trabajo, entre ellos, Lucía Vivancos, Javier Ruíz del Monte y Germán Cuerdo Fernández.

INDICE

PREFACIO

A un conjunto heterogéneo formado por doce personas de edades comprendidas entre los diez y los ochenta y seis años, de distinta formación académica, que viven en lugares diferentes, se les ha pedido que dibujen, sin utilizar símbolos alfabéticos ni numéricos, en el orden dado, y en un intervalo de tiempo limitado que dependía de la edad de la persona, los siguientes conceptos, entre otros, con el máximo nivel de detalle:

* *Bombón*
* *Tiempo*
* *Alaska*
* *Metro*

Se observó que la visión que cada uno de los geniales pintores plasmaron en sus dibujos venía condicionada por diversas cuestiones, entre otras:

- el conocimiento subjetivo que cada uno de ellos poseía del concepto en cuestión, y
- la capacidad que cada uno de ellos tenía para expresarse en un lenguaje distinto al lenguaje habitual, pero conocido por todos.

Así, hubo conceptos, como el concepto *Metro Cuadrado*, que algunos fueron incapaces de dibujar por no poseer el conocimiento y otros por no saber expresarlo. Otros, como el concepto *Metro*, se entendía perfectamente, pero, dependiendo del lugar en el que habitaba la persona, este concepto tenía distinto significado. Y, por último, otros conceptos como el de *Tiempo*, que todos conocían pero que para la gran mayoría era difícil de expresar. El resultado fue un conjunto de dibujos diferentes asociados a cada uno de los conceptos anteriormente mencionados, de los cuales, los más significativos y representativos se han reproducido, siempre respetando la genialidad del autor, en este prefacio. A todos ellos y a quien los ha reproducido, mi más sincero agradecimiento.

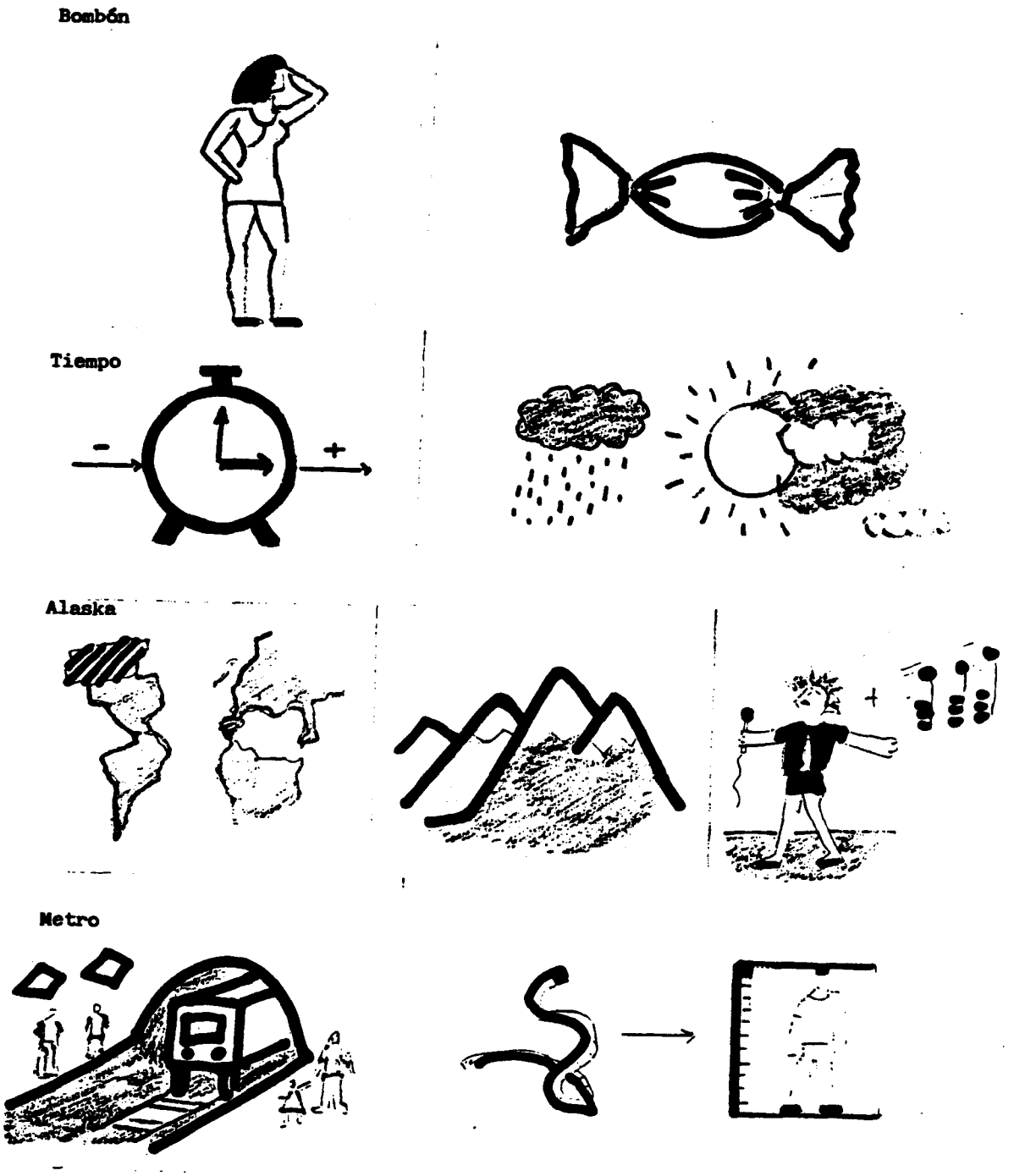


Figura I. Distintas representaciones de varios conceptos

Las consecuencias inmediatas de este pequeño experimento son:

1. Un mismo concepto, o conocimiento, puede representarse de múltiples formas. Esto lleva a pensar que alguna de estas formas es más apropiada que otras.

2. La representación del concepto no siempre refleja la realidad, refleja la realidad del agente. Esto puede provocar mal entendidos cuando la representación es utilizada como medio de comunicación entre dos entidades.

3. Dada la misma representación de un concepto para dos agentes, la habilidad en el uso del lenguaje influye drásticamente en la eficacia de la acción.

Pues bien, en el ámbito de la Informática, y más concretamente de la Inteligencia Artificial y su pretensión de dotar a las computadoras de inteligencia y por tanto de que sean capaces de manejar conocimiento, el problema de la representación y el uso de los lenguajes o Formalismos también existe, y, quizás, no es exagerado decir que este problema se encuentra especialmente agudizado.

1. INTRODUCCION

1. INTRODUCCION

No se equivocaba A. Turing [Turing, 50] cuando, en 1950, afirmaba:

> *Creo que al final de este siglo, nuestras ideas y opiniones habrán evolucionado lo suficiente como para poder hablar sin rubor de máquinas pensantes.*

En dicha comunicación Turing proponía una prueba basada en el juego de la imitación que prentendía evaluar en qué medida las respuestas de las máquinas pensantes se asemejaban a las respuestas de un ser inteligente. Desde entonces, muchas medidas se han desarrollado en este sentido. El Test de Turing [Turing, 50], la Serendipia y el Holismo [Roberts, 92], [Caraça et al., 93] son alguna de ellas. En definitiva, lo que los distintos criterios de evaluación pretenden es calificar en qué medida las máquinas proporcionan respuestas similares a las que podría dar un ser humano inteligente para, de este modo, decidir si pueden ser consideradas inteligentes o no. Básicamente, se distinguen tres tipos de respuestas.

1. Si las respuestas dadas por la máquina son más inteligentes que las del humano, entonces, al haber construido una máquina más inteligente que el hombre, el objetivo se ha cumplido.

2. Las respuestas proporcionadas por la máquina son similares a las del humano inteligente. En este caso, se dice que las máquinas se comportan inteligentemente, al igual que lo hace el humano. También aquí, el objetivo se ha cumplido.

3. Finalmente, las respuestas dadas por la máquina son menos inteligentes que las proporcionadas por el humano. Entonces, hay que analizar el factor desencadenante que ha originado que las respuestas sean diferentes. En el análisis de las causas que lo han podido ocasionar, se distinguen tres casos.

   3.1 Primero, que el conocimiento almacenado en la computadora no coincida con el conocimiento del humano. En este caso, se debe modificar el conocimiento almacenado hasta reflejar el conocimiento que de la realidad posee el hombre. Una vez modificado, se vuelven a analizar las respuestas.

3.2 Segundo, que los mecanismos de inferencia que razonan con el conocimiento, supuesto éste válido, son incorrectos. Bien porque las inferencias estén mal implementadas y razonan mal con el conocimiento almacenado, bien porque, aún estando bien implementadas, producen nuevo conocimiento erróneo, o bien, porque las inferencias no se parecen en nada a las realizadas por el humano. En cualquier caso, se deben modificar las técnicas de inferencia y proporcionar unas correctas que sean lo más parecidas posible al razonamiento realizado por el hombre. Una vez modificadas, se vuelven a comparar las respuestas.

3.3 Y tercero, que aún coincidiendo el conocimiento de la computadora con el conocimiento del humano y estando las inferencias bien realizadas, se hayan perdido cantidades significativas de conocimiento en cada una de las fases por las que se ha pasado al construir el sistema inteligente. En palabras de Newell [Newell, 82], *el conocimiento del mundo no puede capturarse en una estructura finita.* El conocimiento de la realidad que no se ha introducido en la Base de Conocimiento (BC) del sistema origina incompletud en los datos en ella almacenados y, por consiguiente, incertidumbre en los resultados al aplicar técnicas de inferencias válidas. En este caso, es necesario proporcionar mayor expresividad a la *estructura finita* con el fin de incorporar más información al sistema e introducir o modificar técnicas de inferencia que trabajen eficientemente con los nuevos elementos incorporados en ella. Este trabajo se centra en este último supuesto dado que es el único que está aún sin resolver.

Las pérdidas de conocimiento no se producen de repente en los sistemas inteligentes, sino que se van produciendo significativa y progresivamente a medida que éste se va desarrollando a lo largo de las siguientes etapas:

Etapa 1. Selección del experto humano.

La concepción que distintos humanos tienen de la realidad o *Mundo Externo* en un dominio es una visión subjetiva, propia y diferente de la de los demás. Aunque esta etapa no va a ser objeto de estudio en este trabajo, parece lógico elegir aquel humano

cuyo *Modelo Subjetivo* de la realidad se asemeje más al *Mundo Externo*. Así, el Ingeniero del Conocimiento (IC) que debe construir el sistema inteligente, tendrá asegurado que las distorsiones, ruidos o desviaciones del *Modelo* que él construirá serán siempre las mínimas. El problema, entonces, se encuentra en averiguar qué *Modelo Subjetivo* se asemeja más a la realidad si, como ocurre en la mayoría de los casos, el *Mundo Externo* es parcialmente conocido.

Etapa 2. Adquisición del Conocimiento y Conceptualización.

Una vez seleccionado el experto humano y, por tanto, el *Modelo Subjetivo* de la realidad, comenzará la Adquisición del Conocimiento y la Conceptualización del mismo. Estas etapas tampoco entran dentro del alcance de este trabajo, a pesar de que en ellas también se produce una pérdida de conocimiento, bien al no adquirir todo el conocimiento del experto humano, bien al no conceptualizar todo el conocimiento adquirido. En la etapa de conceptualización se describe el conocimiento independientemente de cómo éste se almacena en la computadora. Pionero en la separación de la descripción del conocimiento de su representación fue Newell [Newell, 82], al proponer el llamado Nivel de Conocimiento sobre los niveles tradicionales de una computadora (Simbólico, Lógico, de Circuito y de Dispositivo). Entre las propuestas de estructuración del conocimiento en este nivel cabe destacar la de KADS [Breuker et al., 85], los Componentes de la Experiencia [Steels, 90] y la Unidad Cognitiva [Molina, 93]. Todas ellas pretenden producir *Modelos Cognoscitivos* del *Modelo Subjetivo* del experto

Etapa 3. Formalización.

Realizada la descripción del sistema en el Nivel de Conocimiento, es necesario expresar el conocimiento en el Nivel Simbólico. En este sentido, los formalismos de representación del conocimiento permiten expresar el conocimiento del *Modelo Cognoscitivo* en esquemas formales o en *Modelos Formalizados* comprensibles por la computadora. El IC debe seleccionar, como paradigma de representación del conocimiento, aquel que mejor permita expresar, de manera natural y completa, el conocimiento del mundo. Cuanto más completo o fiel sea el esquema a la realidad, mejor trabajará la computadora con él y mejores serán los resultados, haciéndole más inteligente. Esta etapa sí se trata en este trabajo.

Etapa 4. Implementación.

Una vez formalizado el conocimiento, dicho conocimiento y sus técnicas de inferencia asociadas, deben introducirse en una computadora, bien utilizando entornos comerciales o bien programando a medida el sistema completo. En el primer caso, cada entorno poseerá un *Modelo Interno* implementado que puede parecerse, o no, al formalismo de representación elegido. En el segundo, habrá que crear el *Modelo Interno* y, por consiguiente, habrá que elegir el lenguaje de programación que mejor permita expresarlo. En ambos casos, existen múltiples herramientas y lenguajes de programación sobre los cuales se puede realizar esta tarea. El IC deberá seleccionar aquella herramienta o aquél lenguaje que, de manera natural permita expresar la técnica empleada en la formalización del conocimiento y sus inferencias asociadas. El *Modelo Interno* así construido debería permitir expresar de manera natural todo el conocimiento formalizado.

La figura 1.1 muestra las distintas percepciones o *Modelos Subjetivos* que de la realidad o *Mundo Externo* tiene cada experto; distintos *Modelos Cognoscitivos* asociados a cada *Modelo Subjetivo;* distintas formalizaciones de cada *Modelo Cognoscitivo;* y cómo cada una de estas formalizaciones pueden implementarse en la computadora utilizando diversos *Modelos Internos*. Como puede verse en la figura, son muchos los caminos que conectan el *Mundo Externo* con los distintos *Modelos Internos* que lo implementan. Se puede intuir que, dependiendo del camino elegido, se llegará a un rotundo éxito, a un fracaso estrepitoso o a soluciones intermedias aceptables. El primer estado, se alcanzará si las respuestas proporcionadas por la máquina coinciden con las del experto; el segundo, si las respuestas nunca son las mismas; y, el tercero, si en ocasiones acierta y en otras falla, pero, en general, son respuestas aceptables. La diferencia en las respuestas dadas por el sistema y por el experto, tomando siempre como referencia la figura 1.1, se deben a que:

1. La técnica de representación elegida no es la adecuada. Es decir, el formalismo seleccionado no se adecúa bien a la forma en la que el conocimiento está presente en la realidad. Normalmente, se debe a que la expresividad del formalismo es limitada. En este caso, o se incorporan al formalismo elementos que incrementen su expresividad o se selecciona otra técnica de representación que se adecúe más al conocimiento del dominio. Este trabajo profundiza en esta línea al incorporar elementos que incrementan la expresividad de un formalismo.

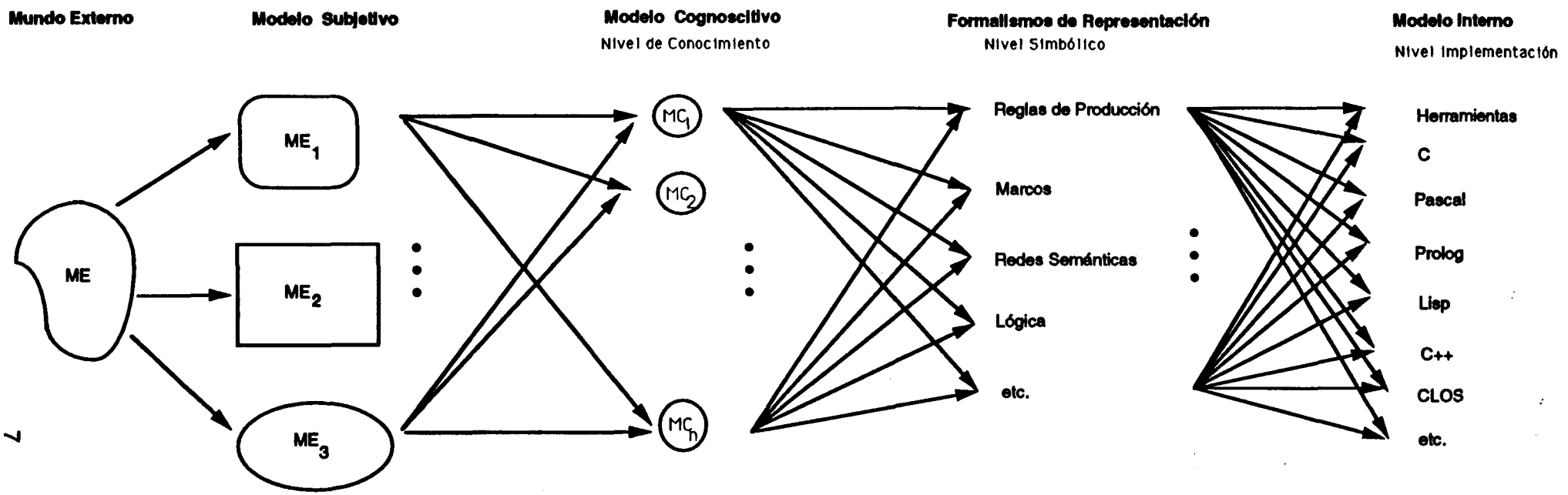


Figura 1.1. Múltiples Modelos Internos para un único Modelo Externo

2. No todos los formalismos tienen asociados técnicas de inferencia que utilizan el conocimiento de forma similar a cómo éste se utiliza en la realidad. Si esto ocurre, se deberán incorporar al formalismo nuevas técnicas de inferencia o modificar las ya existentes. Este trabajo profundiza también en ambas líneas.

3. Aún siendo el formalismo y sus inferencias asociadas adecuadas, la implementación del mismo no es correcta. En este caso, habrá que modificar la implementación del sistema. En este trabajo, al crear un entorno que ayuda al IC a formalizar bases de conocimientos, se evita la implementación del sistema final y, por consiguiente, este tipo de problemas.

4. La implementación no refleja de manera natural el conocimiento ya formalizado. Bien porque la herramienta o el lenguaje de programación seleccionado no facilita la implementación del conocimiento formalizado en un *Modelo Interno*, o porque es difícil simular la inferencia en la herramienta o en el lenguaje elegido. En este caso, habrá que cambiar de herramienta o de lenguaje de programación. Este problema no se tratará en este trabajo.

5. Por último, algunas técnicas de representación han evolucionado "a posteriori" y al compás de las herramientas comerciales que implementan el formalismo. Así, las distintas herramientas han introducido nuevos conceptos en los formalismos y cada una los ha utilizado de una determinada manera. Los conceptos que se han revelado más útiles se han incorporado al formalismo y aquellos que no lo han sido se han desechado. Por ello, se puede hablar de un desfase teórico existente entre los formalismos de representación y las herramientas que lo implementan. Lenzerini, Nardi y Simi [Lenzerini et al., 91] ponen de manifiesto el desfase existente entre las herramientas que implementan el conocimiento ya formalizado y los formalismos de representación, y proponen, como línea de investigación, disminuir este desfase. Este trabajo profundiza también en esta línea.

Todo ello pone de manifiesto que es necesario acercar el *Modelo Formalizado* al *Modelo Cognoscitivo* con el fin de disminuir la pérdida de conocimiento que se produce

al formalizar, y que es necesario acercar el *Modelo Interno* al *Modelo Formalizado* si se quiere disminuir la pérdida de conocimiento que se produce al implementar. Blum [Blum, 92] expresa este proceso, en el desarrollo del Software convencional, como se muestra en la figura 1.2, y expone, que para cada tipo de Modelo Conceptual existen múltiples Modelos Formales, y que, para cada Modelo Formal, existen numerosas implementaciones correctas. El Ingeniero del Software seleccionará, basándose en su juicio y en su experiencia, el Modelo Conceptual y el Modelo Formal que mejor resuelva su problema.

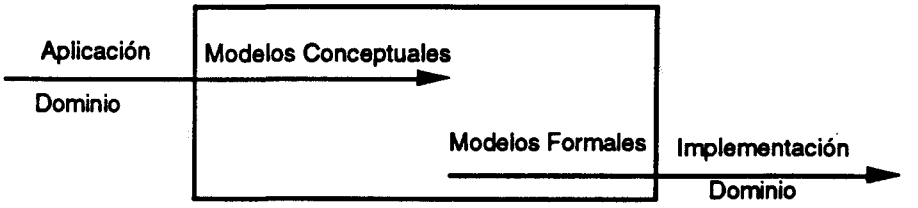


Figura 1.2. El Modelo Conceptual y el Modelo Formal en la Ingeniería del Software

Trasladando la opinión de Blum a la Ingeniería del Conocimiento, el Ingeniero del Conocimiento seleccionará el *Modelo Formalizado* que mejor represente el conocimiento del *Modelo Cognoscitivo* y, una vez seleccionado dicho *Modelo*, seleccionará el *Modelo Interno* que mejor implemente el conocimiento formalizado y sus técnicas de inferencia. De esta forma, se asegura que las pérdidas de conocimiento que se producen en el proceso de construcción del sistema inteligente son mínimas.

Desafortunadamente, no existen técnicas de representación del conocimiento que optimicen todas las características expuestas por Davis [Davis et al., 93] y por Rich [Rich et al., 91] que son deseables en los formalismos. El resultado, es un conjunto de técnicas diferentes de representación. Actualmente, dada la expresividad y eficacia notacional de los Marcos, éstos se han convertido en uno de los Paradigmas de Representación del Conocimiento más utilizado en Inteligencia Artificial (IA). Este formalismo es adecuado cuando la realidad está organizada en base a conceptos o entidades, clasificaciones o taxonomías de estos conceptos, propiedades y relaciones.

Actualmente, existe una gran variedad de herramientas comerciales que facilitan la labor del IC en la implementación del sistema ya formalizado en Marcos. Sin embargo, el IC sigue formalizando manualmente y transformando también manualmente el conocimiento formalizado al Modelo Interno que la herramienta proporciona. Además de utilizar estas herramientas para implementar Sistemas Basados en Marcos, también se pueden utilizar lenguajes específicos de programación de alto nivel como Pascal o C; lenguajes funcionales como Lisp; lenguajes de

programación lógica como Prolog; o bien, lenguajes de programación orientados a objetos (POO) como C++, CLOS *(Common Lisp Object System)*, Eiffel, etc. Dadas las similitudes, al menos en la superficie, entre Marcos y Objetos, la POO se ha revelado cono una técnica de programación adecuada para implementar Sistemas Basados en Marcos y, por consiguiente, para crear *Modelos Internos* del *Modelo Formalizado* construido por el IC.

Según todo lo dicho anteriormente, se propone en este trabajo un **Modelo de Diseño Orientado a Marcos** que, intermedio entre el *Mundo Externo* y el *Modelo Cognoscitivo*, disminuya las pérdidas de conocimiento que se producen al construir sistemas inteligentes.

Por otro lado, y como aplicación práctica del Modelo de Diseño Orientado a Marcos, se construye un entorno que permite construir cualquier Sistema Basado en el Conocimiento (SBC) que utilice como técnica de representación el formalismo de Marcos. Este entorno se fundamenta en el Modelo de Diseño y es un SBC basado en Marcos que ha sido programado con orientación a objetos. Se trata, por tanto, de una cierta anidación. Es decir, se trata de marcos que permiten formalizar cualquier SBC en marcos. Con él se consigue:

a) Que el IC formalice automáticamente el sistema. Esto quiere decir que el entorno, utilizando el Sistema Basado en Marcos (SBM) sobre el cual está construido, y, razonando con el conocimiento de la aplicación, podrá inferir nuevo conocimiento y razonar con el conocimiento incierto presente en el dominio. Además, el IC podrá validar el conocimiento del dominio en la fase de formalización, en lugar de tener que esperar a que el sistema esté implementado.

b) Que una vez formalizado todo el conocimiento en marcos, el IC no tenga que implementar el sistema, pues el entorno permitirá al usuario final trabajar con el conocimiento en él almacenado utilizando las inferencias en él permitidas.

2. ESTADO DE LA CUESTION

d) Además, cualquier operando presente en cualquier operación aritmética será del tipo *Operaciones_Aritméticas*, permitiéndose así la construcción de expresiones aritméticas complejas e impidiendo el uso de expresiones lógicas dentro de ellas.

II.3.2.2 Resultado de la Operación.

A diferencia de lo que sucedía en las *Operaciones_Lógicas*, en las *Operaciones_Aritméticas* el *Resultado* de la operación depende del tipo de operación que se está ejecutando. Por ejemplo, el resultado de la operación división entera es un entero mientras que el resultado de la operación división es un real. Por este motivo, el *Resultado* de la evaluación de la operación no se va a definir globalmente en el Marco *Operación_Aritmética*, sino que se va a definir localmente en los Marcos que representan los tipos de operaciones, siendo la ranura *Resultado* del mismo tipo que el de los operandos allí definidos.

II.3.3 EJEMPLO DE EJECUCION DE UNA OPERACION

Antes de pasar a describir la representación de una operación compleja mediante marcos, se va a exponer cómo se realiza la ejecución de una operación en el SBM. Cada marco clase de la jerarquía de operaciones aritméticas y lógicas incluye una ranura llamada *Ejecutar*, definida de tipo *método*, que almacena el nombre de un procedimiento y los parámetros con que tiene que ejecutarse cuando el marco reciba una señal de activación de otro marco o del exterior. El procedimiento aquí definido se corresponde con un método perteneciente a una clase de la jerarquía de objetos que representan los tipos básicos en C++.

Por ejemplo, cuando se solicite la ejecución de una operación *Y* concreta, se enviará un mensaje a un marco instanciado instancia del marco clase *Operador_Y*. Al recibir el mensaje de activación, el marco instanciado buscará si en él se ha rellenado la ranura *Ejecutar*. Como en él no se ha definido la ranura, aplicando herencia de propiedades, buscará la ranura en los marcos con los que está conectado en la jerarquía. Una vez encontrada la ranura pueden suceder dos cosas:

a) Si alguno de los dos operandos que forman la operación es un operando compuesto en el que interviene una operación, se enviará un mensaje al marco instanciado que representa la operación.

2.1. LA REPRESENTACION DEL CONOCIMIENTO

2.1.1. LA INTELIGENCIA ARTIFICIAL Y LA REPRESENTACION DEL CONOCIMIENTO

La mayoría de los programas que trabajan en Inteligencia Artificial (IA) manipulan símbolos, que representan piezas de información sobre el mundo, con el fin de realizar unas tareas a las que normalmente se les pone el calificativo de inteligentes. Para que una máquina se comporte de forma inteligente debe poseer conocimiento y debe ser capaz de utilizarlo. No basta con introducir el conocimiento en la máquina, hay que proveerla de unos mecanismos que le permitan razonar con el conocimiento previamente almacenado. La representación y el razonamiento, por consiguiente, están estrechamente relacionados de tal forma que no se puede hablar de una sin hablar del otro, y viceversa.

El área de la IA que estudia cómo representar el conocimiento adquirido del mundo real en una computadora, se llama *Representación del Conocimiento*. A las distintas estructuras que permiten expresar conocimiento y a los mecanismos que lo utilizan se les llama *Formalismos* o *Técnicas* de Representación del Conocimiento [Davis et al., 93]. Para Brachman y Levesque [Brachman et al., 85] la Representación del Conocimiento consiste en realizar descripciones del mundo de tal forma que una máquina pueda establecer nuevas conclusiones sobre su entorno manipulando dichas descripciones. Por tanto, el proceso de Representación del Conocimiento implica traducir el conocimiento de un dominio en un formalismo de representación que pueda ser utilizado por una computadora para generar comportamientos inteligentes.

Aunque existen múltiples formas de realizar programas inteligentes, la mayoría han tomado como punto de partida la *Hipótesis de Representación del Conocimiento* de Brian Smith [Smith, 82]. Dicha hipótesis enuncia que cualquier sistema que presente un comportamiento más o menos inteligente, utiliza una estructura para representar el conocimiento, que recibe el nombre de Base de Conocimientos (BC), y un Motor de Inferencia, separado de la BC, que lo manipula. De esta forma, cada formalismo de representación, tendrá unas formas concretas de realizar la inferencia que le permitirán obtener nueva información a partir de la información explícita almacenada en la BC. Siempre en la Hipótesis de Representación del Conocimiento, los procedimientos del Motor de Inferencia son completamente

independientes del conocimiento que está almacenado en la base, pero no de la forma en que éste se almacena.

Enfrentada con esta hipótesis de representación se encuentra la *Hipótesis Conexionista.* Esta segunda hipótesis rechaza la idea de módulos separados y defiende que la información se debe almacenar en la computadora de forma similar a cómo el ser humano la almacena en su cerebro. Es decir, en una red de neuronas el conocimiento se encuentra almacenado en los pesos asociados a las conexiones entre las neuronas de la red, y, las inferencias se realizan mediante señales de activación lanzadas por neuronas que se combinan con los pesos de la red según unas determinadas reglas de propagación que modifican el estado de otras neuronas.

2.1.2. MODELIZACION DEL MUNDO EXTERNO

Cualquier método de resolución de un problema en IA, exige, necesariamente, que se pueda representar el problema de tal forma que pueda ser captado por la máquina [Borrajo et al., 93]. El programa no trata con el mundo físico, sino con una representación o modelización del mundo o del problema. En ocasiones, la traducción de la realidad en términos de una representación es obvia, en otras, puede ser compleja. En cualquier caso, una buena representación sugiere un buen método para resolver un problema.

El ciclo básico que realiza cualquier sistema inteligente que utiliza conocimiento con el fin de alcanzar unos objetivos, podría describirse como se muestra en la figura 2.1 [Woods, 91].

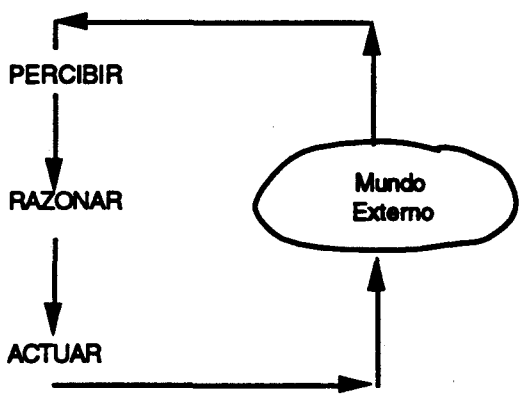


Figura 2.1. Bucle de Razonamiento de un Sistema Inteligente

Así, el sistema está percibiendo contínuamente cosas, razonando sobre ellas y realizando acciones. La principal dificultad de estos sistemas es obtener un *Modelo Interno*, sobre el cual razonar, que represente correctamente la realidad o el *Mundo Externo*. El *Modelo Interno*, sustituto del *Mundo Externo* en la computadora, es siempre inexacto e imperfecto. Las diferencias entre el *Modelo Interno* y la realidad se deben a que:

a) Se pueden haber producido cambios en el *Mundo Externo* desde que el sistema inteligente supo por última vez de él.

b) La incapacidad del sistema inteligente de aprender, en una cantidad de tiempo razonable, todo lo que se podía conocer de la realidad.

c) Las limitaciones que presentan las Técnicas de Representación del Conocimiento hacen que no se pueda representar todo lo que se conoce de la realidad.

En un SBC, la BC es un *Modelo Interno* de la realidad. El sistema inteligente razona sobre la BC, es decir, sobre el *Modelo Interno*, y nunca sobre el mundo exterior. Para que el sistema inteligente razone, es necesario que:

a) perciba el *Mundo Externo*, y

b) actúe sobre el *Modelo Interno* y sobre el *Mundo Externo* con el fin de comparar los efectos que produce la acción, tanto en el *Modelo Interno* como en la realidad. Si los efectos son los mismos, entonces, la acción realizada es correcta. Además, el sistema debe prepararse para una nueva percepción. El proceso descrito por Woods [Woods, 91] se podría expresar gráficamente como se muestra en la figura 2.2.

En definitiva, la clave de la Representación del Conocimiento, está en que el *Modelo Interno* que se ha implementado en la computadora sea lo más parecido posible al *Mundo Externo*. Si lo es, las respuestas y los comportamientos del sistema inteligente serán lo más parecidos posible a las respuestas y al comportamiento que se produce en la realidad y las interacciones entre ambos mundos se realizarían de manera natural.

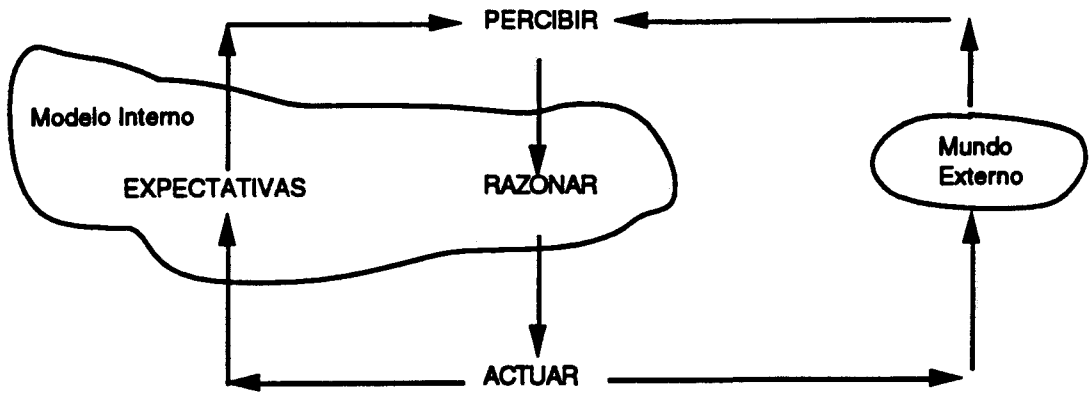


Figura 2.2. Modelización del Mundo Externo

2.1.3 CONOCIMIENTO Y FORMALISMOS DE REPRESENTACION

La adecuación entre el *Modelo Interno*, implementado en la computadora, y la realidad depende, en gran medida, del Formalismo de Representación del Conocimiento seleccionado y de sus limitaciones.

En esta sección se analiza cómo, dependiendo del tipo de conocimiento presente en el dominio, se deben utilizar los distintos formalismos. La selección del Formalismo que mejor represente el conocimiento del *Mundo Externo* y la formalización correcta de la BC utilizando la técnica elegida, permitirá construir un *Modelo Formalizado* asociado al *Mundo Externo* en el que las pérdidas de conocimiento sean mínimas. Con este fin, se describen las características que cualquier Formalismo de Representación debe tener para asegurar la adecuación del formalismo al conocimiento de la realidad. Posteriormente, se analizan diversas manifestaciones del conocimiento en el mundo y los formalismos que mejor representan cada uno de ellos.

2.1.3.1 Características de un Formalismo de Representación

Davis, Shrobe y Szlovits [Davis et al., 93] establecen cinco características que debe cumplir cualquier Formalismo o Técnica de Representación del Conocimiento para poder ser utilizada en la construcción de sistemas inteligentes.

a) La Representación del Conocimiento debe ser un Sustituto Fiel del Mundo Externo

El razonamiento realizado por cualquier sistema inteligente que interacciona con el *Mundo Externo* es un proceso interno que se realiza en una computadora sobre cosas que solamente existen en la realidad. La Representación del Conocimiento, al hacer las veces de mundo externo en el interior del sistema inteligente, es el sustituto del *Mundo Externo.* Dado que existen múltiples sustitutos de la realidad, habrá sustitutos que se adapten mejor que otros a ella. Siempre, se debe elegir aquél sustituto o representación que minimice las diferencias y los errores. Cualidades como fidelidad y exactitud con el *Mundo Externo*, son necesarias en el *Modelo Interno.*

b) Una Representación del Conocimiento debe ser un Filtro adecuado del Mundo Externo

La selección de un formalismo de representación obliga a representar el mundo de una determinada manera, centrándose en los aspectos relevantes y obviando los que no lo son. Es decir, es como si el formalismo colocase unas gafas determinadas para ver el mundo. Las gafas de la técnica empleada condicionan qué y cómo ver el mundo, qué elementos atender y cuáles ignorar.

El problema o problemas a tratar en el *Mundo Externo* también imponen la atención o ignorancia de ciertos elementos del mundo. En este sentido, se atenderán aquellos aspectos que contribuyan o participen en el problema y su solución. Se ignorarán aquellos elementos del *Mundo Externo* que no influyan en el problema a tratar. La adecuación entre el filtro, que supone la Técnica de Representación, y el problema a tratar es fundamental para conseguir la inteligencia del sistema.

c) El Razonamiento sobre el Mundo Real debe ser Inteligente

Toda técnica de representación lleva asociada una teoría que permite razonar inteligentemente con ella. El formalismo elegido debe permitir que el sistema sustente un comportamiento inteligente. Para ello se debe, en cada contexto:

1. Llegar a un acuerdo en el significado de la palabra "Razonamiento Inteligente".

2. Asegurar que el sistema no realice razonamientos erróneos.

3. Asegurar que el sistema realice todos los posibles razonamientos correctos en el contexto o dominio dado.

d) La Representación del Conocimiento debe ser un Medio de Computación Eficiente

Además de una modelización de la realidad, la Representación elegida dentro de la máquina es un medio de computación. La eficiencia computacional del Formalismo de Representación del Conocimiento es fundamental. Es decir, la organización de la información en la computadora debe hacerse de tal forma que facilite la realización de las inferencias. Si no las facilita, la representación no es eficiente, y por tanto, tampoco es útil.

e) La Representación del Conocimiento debe ser un Medio de Expresión Fluido

Las técnicas de Representación del Conocimiento también pueden verse como un medio de expresión utilizado por el hombre para decirle a la máquina cómo es el mundo. Por tanto, cualquier formalismo de representación debe permitir una comunicación fluida, sin grandes esfuerzos, y precisa, entre el hombre y la máquina.

Otros autores comentan otros aspectos como: la exactitud [Borrajo et al., 93], la utilidad [Borrajo et al., 93], y la notación [Woods, 91] de la representación. La **exactitud** se refiere a si el modelo utilizado es lo suficientemente fiel a la realidad como para no distorsionarla. En este sentido, esta característica es semejante a la primera característica enunciada por Davis. La **utilidad** se refiere a si el programa usa fácilmente el formalismo. En este caso, la característica cuarta de Davis tiene contenido semejante. Y, por último, la **notación** utilizada en la Representación del Conocimiento es muy importante computacional y conceptualmente, que viene a resumir las características primera, segunda y cuarta de Davis. La adecuación de la notación utilizada en la Representación del Conocimiento trata el poder expresivo y la eficacia de la notación.

* El *Poder Expresivo* se refiere a qué se puede decir con la notación. Es decir, si la representación permite realizar o evitar distinciones sutiles.

* La *Eficacia de la Notación* se refiere a las estructuras concretas que se utilizan y al impacto que estas estructuras tienen en las operaciones del sistema. Una notación es eficaz si es computacionalmente eficiente, conceptualmente clara, concisa y facilita las modificaciones.

Para Rich y Knight [Rich et al., 91], las propiedades deseables en un formalismo de representación son:

1. Adecuación de la representación:

   Se refiere a la habilidad de representar todo el conocimiento presente en el dominio. Esta característica es semejante al poder expresivo en la notación de Woods considerada.

2. Adecuación a las inferencias:

   Se refiere a la habilidad de manipular estructuras de conocimiento, de tal forma que se infieren nuevas estructuras a partir de estructuras ya existentes en la BC. En este sentido, la característica cuarta de Davis y la utilidad son criterios parecidos.

3. Eficiencia "inferencial":

   Se refiere a la capacidad de incorporar en las estructuras de conocimiento información adicional que sea utilizada para focalizar la atención de los mecanismos de inferencias. Esta característica no ha sido contemplada por los autores anteriores.

4. Eficiencia en la adquisición de nueva información:

   Se refiere a la capacidad que tiene el sistema de adquirir nueva información y representarla.

Por tanto, como puede observarse, las características esenciales y necesarias que deben estar presentes en una Técnica de Representación del Conocimiento son: la fidelidad, exactitud o adecuación al *Mundo Externo*, y la eficiencia computacional de la

Representación dentro de la máquina. Desafortunadamente, no existen sistemas que optimicen todas las capacidades. El resultado, es un amplio conjunto de formalismos de representación del conocimiento.

2.1.3.2 Tipos de Formalismos

Analizadas las propiedades que deben estar presentes en cualquier sistema de representación, se pasan a analizar las técnicas más utilizadas. Dado que todas las representaciones son imperfectas, se debe elegir aquella que mejor se adecúe a cada caso. Una de las clasificaciones se basa en determinar si el formalismo es más adecuado para representar conceptos, relaciones o acciones.

a) Formalismos basados en Conceptos.

Los formalismos basados en conceptos representan las principales entidades del dominio, sus propiedades y los posibles valores que puede tomar cada propiedad. Los principales formalismos basados en conceptos son: Objeto-Atributo-Valor y los Marcos.

El formalismo **Objeto-Atributo-Valor** es el formalismo más sencillo que permite describir conceptos. El primer elemento, el objeto, describe la entidad o concepto, por ejemplo, una persona. El segundo elemento, el atributo, describe una característica o propiedad de la entidad, por ejemplo, la profesión. El tercer elemento, el valor, describe el valor del atributo del objeto, por ejemplo, abogado. Una BC construida utilizando este formalismo tendrá, para cada objeto, tantas ternas como atributos distintos se quieran representar. Si un atributo puede tomar más de un valor, éste aparecerá tantas veces como valores posibles pueda tomar.

El formalismo de **Marcos**, en inglés *Frames*, fué definido por Minsky [Minsky, 75] como una estructura de datos que representa situaciones estereotipadas. Un marco es una estructura de datos formada por un nombre y un conjunto de propiedades llamadas ranuras, en inglés *slots*. Cada ranura tiene unas propiedades, llamadas facetas, que describen el tipo de valores que puede tomar. Los marcos se relacionan con otros marcos formando una jerarquía de marcos. Así un marco que representa un concepto se relaciona con otros marcos que son los subconceptos del marco que origina la clasificación.

Son varias las diferencias que se pueden encontrar entre el formalismo de Marcos y el formalismo Objeto-Atributo-Valor. La primera, es que los marcos representan el conocimiento del mundo de forma organizada, y la segunda, es que los marcos pueden definir los valores de las propiedades utilizando conocimiento declarativo y procedimental, mientras que el segundo sólamente lo puede hacer utilizando conocimiento declarativo.

b) Formalismos basados en Relaciones.

Los formalismos basados en relaciones focalizan su atención en las relaciones que aparecen entre los conceptos o entidades del dominio. Los más importantes son la Lógica, las Redes Semánticas y la Teoría de la Dependencia Conceptual.

Aunque existen muchas clases de lógica, la más utilizada en los Sistemas Basados en el Conocimiento es el **Cálculo de Predicados de Primer Orden** (CPPO). En este formalismo, las relaciones se representan por predicados y los conceptos son sus argumentos. Estos predicados pueden verse afectados por las conectivas y cuantificadores del CPPO.

Las **Redes Semánticas** fueron definidas por Quillian [Quillian, 68]. Una red semántica es un grafo orientado formado por nodos y por arcos unidireccionales. Los nodos representan conceptos y los arcos representan relaciones entre conceptos. Ambos, nodos y arcos, están etiquetados por el usuario.

La **Teoría de la Dependencia Conceptual** fue propuesta por Schank [Schank, 72], [Schank, 75], [Schank et al., 77]. Esta Teoría proporciona una estructura de representación del conocimiento mediante un conjunto de primitivas conceptuales, categorías conceptuales, relaciones conceptuales y estados. Se utiliza para expresar, de forma declarativa, las relaciones entre los elementos de una frase expresada en lenguaje natural y, para mostrar relaciones entre entidades y objetos del dominio. La gran ventaja de esta teoría frente a los formalismos anteriores es que proporciona al IC un conjunto de primitivas independientes del idioma en el que él se expresa. La única forma que tiene el IC de definir las relaciones en el dominio es utilizando estas primitivas.

c) Formalismos basados en Acciones.

Los formalismos basados en acciones describen el conocimiento del dominio como un conjunto de acciones básicas. Los principales formalismos son las Reglas de Producción y los Guiones.

Las Reglas de Producción presentan la forma SI-ENTONCES. A la parte SI de la regla se le llama antecedente y a la parte ENTONCES se la llama consecuente. Cuando todos los antecedentes de la regla son ciertos se tienen todos los consecuentes.

Los Guiones [Schank et al., 77] describen una secuencia de sucesos estereotipados que tienen lugar en un contexto. Los guiones se ajustan bien para tratar situaciones dinámicas.

2.1.4 COMENTARIOS

En base a todo lo dicho, cabe concluir que:

1. Antes de elegir el formalismo de representación del conocimiento que origine el *Modelo Formalizado*, modelo intermedio entre el *Modelo Interno* y el *Mundo Externo*, se deben estudiar las características que presenta el conocimiento del mundo externo y, en función de ellas, seleccionar el formalismo que mejor lo represente. Otros factores, como pueden ser la tarea y el dominio en el cual se va a utilizar el SBC, también influyen.

2. El conocimiento almacenado en el *Modelo Formalizado*, resultado de aplicar un formalismo de representación, debe ser lo más parecido al conocimiento de la realidad.

3. La Representación del Conocimiento y el Razonamiento están estrechamente relacionados. Cualquier formalismo de representación debe llevar asociado unas inferencias que trabajan con él de manera natural y eficiente.

Actualmente dado que no existen sistemas que contengan todas las características deseables en un formalismo de representación, el formalismo más utilizado, por su mayor expresividad y eficacia computacional son los Marcos. Las

características que a continuación se citan, han hecho de los Marcos el formalismo de representación del conocimiento más usado. Estas características son:

* Permiten construir fácilmente grandes BB.CC. al capturar y representar, de forma intuitiva y natural, la estructura del dominio, sus entidades, sus relaciones, y la forma en la que el experto razona con ellas [Fikes et al., 85].

* La forma de describir las entidades en un SBM es por especialización, es decir, comparando la entidad con otras entidades ya conocidas [Fikes et al., 85].

* Las relaciones taxonómicas entre marcos permiten que una misma información sea compartida por varios marcos vía herencia de propiedades [Fikes et al., 85].

* La propia estructura interna de los marcos permite mantener internamente las restricciones de integridad semántica que existen entre los elementos del dominio [Fikes et al., 85].

* Crea taxonomías de conocimiento, facilitando tanto el diseño como el mantenimiento, al reducir la complejidad de las Bases de Conocimiento [Frost, 86].

* Permite el uso de valores por omisión [Frost, 86].

* Permite el uso de propiedades genéricas [Frost, 86].

* Permite utilizar, de forma integrada, el conocimiento declarativo y el conocimiento procedimental [Frost, 86].

* Los marcos se pueden combinar con reglas en la misma aplicación. Los marcos se utilizan para representar el conocimiento y las reglas para realizar inferencias [Fikes et al., 85].

* Un SBM, mediante el uso de demonios, es capaz de especificar acciones que deberían realizarse cuando ciertas circustancias ocurran durante el procesamiento de la información en la BC. Aunque pueda parecer similar a lo que ocurre en los Sistemas Basados en Reglas, la diferencia, y al mismo tiempo la gran ventaja, de los demonios frente a las Reglas está en

que, a diferencia de los Sistemas Basados en Reglas, en los que las Reglas son chequeadas contínuamente para ver si son aplicables, los demonios de un SBM permanecen inactivos o latentes en el marco hasta que son disparados, sólo entonces son ejecutados [González et al., 93].

* El conocimiento en los SBM está más organizado y estructurado que en el resto de los formalismos. Esto facilita el almacenamiento y la recuperación de la información almacenada en la BC e incrementa la velocidad de los procesos de inferencia [González et al., 93].

* Los SBM son autodirigidos. Los propios marcos determinan cuándo son aplicables a diferentes situaciones dadas. Cuando un marco no se puede aplicar, el marco puede sugerir otros marcos aplicables a la situación [González et al., 93].

* Almacena valores dinámicos. Durante la ejecución de un SBC, los valores de las variables pueden ser almacenados dinámicamente en las ranuras de los marcos. Esto es útil cuando se aplican los SBC a simulaciones, planificación, diagnóstico de problemas, etc. [González et al., 93].

2.2 MARCOS

2.2.1 INTRODUCCION

La Teoría de Marcos es un paradigma que permite representar conocimiento en una computadora. El nombre de *Marcos*, aunque dado por Minsky en 1975 [Minsky, 75], procede de ideas previas tales como: Los *Noemata* que fue como Husserl, en 1913, denominó a los estereotipos. Los *Schemata* de Barlett [Barlett, 32] quien, en 1932, en el contexto de la etología, denominó así a dichos estereotipos.

Minsky [Minsky, 75] definió los **Marcos** como *una estructura de datos que representa situaciones estereotipadas* que se construyen sobre situaciones similares ocurridas anteriormente, permitiendo así aplicar a situaciones nuevas el conocimiento de situaciones, eventos y conceptos previos. El conocimiento que se expresa utilizando esta estructura de datos es el conocimiento declarativo del dominio. Además, asociados a cada marco, existen varias clases o tipo de información, que se refieren a: cómo

utilizar el marco, qué se espera que suceda a continuación, así como el conjunto de acciones que se deben realizar tanto si las expectativas se cumplen como si éstas fallan. Este tipo de conocimiento, que se representa utilizando procedimientos, recibe el nombre de conocimiento procedimental. Minsky integra ambos tipos de conocimiento en el concepto de Marco y propone un formalismo de representación del conocimiento que permite a las computadoras llegar a conclusiones similares a las que puede llegar un ser humano.

También, en 1975, T. Winograd [Winograd, 75] estudió las ventajas e inconvenientes de los lenguajes de representación del conocimiento declarativos y procedimentales. Winograd llegó a la conclusión de que las representaciones declarativas y procedimentales eran estrategias alternativas que se debían utilizar de forma conjunta para representar el conocimiento estático y dinámico del mundo y propuso, basándose en la idea de Marco, una estructura de representación que aunaba las estructuras declarativas y procedimentales. Este artículo sería el precursor del primer lenguaje de representación del conocimiento basado en Marcos llamado KRL [Bobrow et al., 77]. Algunos de los lenguajes que siguieron a KRL fueron FRL [Roberts et al., 77], UNITS [Stefik, 79], KL-ONE [Brachman, 79], KRYPTON [Brachman et al., 85], THEO [Mitchell et al., 89] y CYCL [Lenat et al., 90] entre otros.

2.2.2 CONCEPTOS DECLARATIVOS EN LOS MARCOS

Puesto que existen muchas herramientas que utilizan como formalismo de representación los Marcos, y puesto que cada una de ellas utiliza una terminología y una sintaxis propia para expresarlos se ha decidido estudiar las propiedades comunes a todos ellos a nivel conceptual. Dado que múltiples términos se utilizan para definir un mismo concepto y que algunos términos tienen más de un significado, se ha optado por analizar las percepciones que distintos autores tienen sobre el concepto de Marco en el plano de la formalización y sin tener en cuenta aspectos de implementación, dado que los Marcos son un formalismo de representación y no un lenguaje de implementación. A continuación, se muestra el núcleo de conceptos declarativos que aparecen en la construcción de sistemas inteligentes basados en Marcos.

2.2.2.1 Descripción de la Taxonomía: Marcos y Relaciones

Son muchas las definiciones que se han dado para el concepto de Marco. A continuación, se citan alguna de las más relevantes: Frost [Frost, 86] definió los Marcos como *una estructura de datos que representan un tipo de entidad;* Rich [Rich et al., 91] define un Marco como *un conjunto de atributos y sus valores asociados que describen alguna entidad en el mundo;* la definición de Walters [Walters et al., 88] que los define como un *conjunto de información relacionada sobre un tópico*; o la de Reichgelt [Reichgelt, 91] que los define como *estructuras que representan conocimiento sobre aspectos limitados del mundo*; o la de Nado y Fikes [Nado et al., 92] que definen a los Marcos como *una estructura de datos que se utiliza para codificar un conjunto de creencias sobre una entidad en el dominio de discurso*; o la definición dada en el *Standard and Review Manual for Certification in Knowledge Engineering* [Steinheiser, 90] que define a los Marcos como *estructuras de datos tabulares que representan prototipos o eventos* o, como *clases que agrupan a objetos similares y, a partir de ellos, se puede formar subclases o instancias de esta clase.* Algunas herramientas utilizan otros términos para referenciar el concepto de Marco. Por ejemplo, KEE llama a los Marcos "Units" y KL-ONE los llama "Concepts".

Todos los autores, básicamente, distinguen dos tipos de Marcos. Rich [Rich et al., 91] analiza los tipos de Marcos tomando como punto de partida la Teoría de Conjuntos. Desde esta perspectiva, un Marco puede representar una clase (conjunto) o una instancia (elemento de un conjunto). Para Rich, los **Marcos Clase**, en inglés *Class Frame*, se refieren a *conjuntos de entidades individuales que comparten las mismas propiedades funcionales o estructurales.* Concretamente, la información por omisión asociada a una clase puede ser utilizada para inferir valores de propiedades de sus elementos individuales. Para Reichgelt [Reichgelt, 91] y Winston [Winston, 92] un Marco Clase, o simplemente una Clase, es una *descripción de una clase de entidades que existen en el mundo.* Un término que está muy relacionado con el término **Clase** que puede llevar a confusión es el de **Prototipo.** Un **Prototipo** [Nado et al., 92] es una parte de un Marco Clase que describe un miembro cualquiera de dicha clase. El resto de la clase que no es el Prototipo, describe a la clase en sí.

A los **Marcos Instancias**, en inglés *Instance Frame*, Rich [Rich et al., 91] los define como *elementos que pertenecen a una clase*, y Frost [Frost, 86], como *entidades concretas del tipo representado por el Marco Clase.*

Los Marcos no sólo permiten representar conceptos, clases, entidades, o situaciones desde un único punto de vista sino que permiten representarlos desde puntos de vista diferentes, dando lugar a Marcos que reciben el nombre de **Marcos Alternativos**, en inglés *Alternative View Frames* [Frost, 86].

Todos estos Marcos Clase y Marcos Instanciados no son entes aislados dentro de la Base de Conocimientos, sino que están relacionados formando un **Sistema de Marcos**, en inglés *Frame System*. Frost [Frost, 86] define la jerarquía de marcos como una red de nodos y de relaciones organizados en un jerarquía, donde los nodos de los niveles superiores representan conceptos generales y los nodos de niveles inferiores representan instancias de aquellos conceptos. Rich [Rich et al., 91] define un Sistema de Marcos como un conjunto de marcos conectados entre sí, de tal forma que el valor que toma un atributo en un marco es otro marco. Además, los marcos se organizan en taxonomías utilizando las relaciones que a continuación se describen.

La **Relación SubClase**, en inglés *AKO Relation,* es la relación más utilizada en una jerarquía de Marcos por establecer una jerarquía hereditaria entre ellos y definir una vía para la herencia de propiedades. Dados dos marcos *A* y *B*, marcos cualesquiera de una jerarquía, se dice que existe una relación SubClase entre *A* y *B*, si *A* representa un conjunto de entidades que son subconjuntos del conjunto de entidades representadas por *B* [Frost, 86]. A esta relación, Rich y Knight [Rich et al., 91], influenciados por la Teoría de Conjuntos, también la llaman *Subset Relation* y, a su relación inversa, relación que une una clase con sus subclases, *Subclass Relation*. Pensando más en la implementación, algunos autores llaman a las relaciones, nexos. Por ejemplo, Reichgelt [Reichgelt, 91] y Winston [Winston, 92] llaman a la relación directa *SuperClass Link* y *A-kind-of Link* respectivamente, y a la relación inversa *Sub Link* y *Class Link*, respectivamente.

Si se sabe que el marco *A* está unido por una relación *SubClase* con el marco *B*, se sabe que *A* es un subconjunto de *B*, o que *A* es una especialización de *B*, lo que, gráficamente, se representa en la Figura 2.3 como:

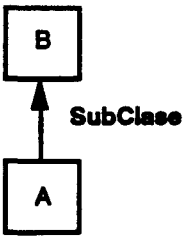


Figura 2.3. Representación de *A* es una SubClase de *B*

Esta relación es distinta a la relación de ser miembro de un conjunto, que matemáticamente se corresponde con la relación pertenece. Si *A* es una clase de *B*, entonces *A* es una especialización de *B*, no un miembro de *B*.

La relación SubClase es muy importante en algunas aplicaciones, puesto que define una vía para la herencia de propiedades. Es decir, si el marco *A* es una subclase de *B* y el marco *B* es una subclase de *C*, entonces, todas las propiedades de la clase definidas en *C* van a ser compartidas con *B* y, por lo tanto, por los elementos y especializaciones de *B*. De la misma manera, todas las propiedades de la clase *B*, incluyendo las heredadas de *C*, se comparten también por *A*, definiéndose en *A* aquellas propiedades que le distinguen de *B*, y en *B* aquellas que le diferencian de *C* [González et al., 93].

En definitiva, independientemente de la terminología empleada, todos los autores coinciden en que esta relación expresa que un conjunto es un subconjunto, una clase o especialización de otro. Por estos motivos, se propone llamar a esta relación **Relación SubClase**, y a su relación inversa **Relación SuperClase.**

La **Relación Instancia**, en inglés *Instance Relation,* entre dos Marcos *A* y *B* se utiliza para expresar que el marco instanciado *A* es una instancia del marco clase *B*. Desde el punto de vista de la Teoría de Conjuntos, Rich [Rich et al., 91], llama a esta relación *Element of* pues representa que una instancia es un elemento de un conjunto, y a su relación inversa le da el nombre de *All-instance*. A esta relación, Frost [Frost, 86] la llama *Set Membership Relationship* y Reichgelt [Reichgelt, 91] *Instance Link*. El enlace inverso al enlace *Instance Link*, Reichgelt lo llama *Member of Link*. Dada la variedad de términos ingleses que se pueden emplear para hacer referencia a esta relación y que, en definitiva, todos expresan que un elemento es instancia de un conjunto, se propone llamar en castellano a esta relación, **Relación Instancia** y, a su inversa, **Relación Elementos_Clase.**

Dos marcos *A* y *B* pueden relacionarse mediante una **Relación Fraternal**, en inglés *Sibling Relationship*, si tanto *A* como *B* tienen como padre el mismo marco *C* [Frost, 86], es decir, si desde *A* y desde *B* parten relaciones SubClase hacia *C*.

Entre dos marcos *A* y *B* existe una **Relación Disjunta**, en inglés *Disjoint Relationship* [Frost, 86], si tanto *A* como *B* representan conjunto disjuntos de entidades. Dos marcos disjuntos nunca se pueden instanciar simultáneamente en marcos alternativos. A esta relación, Rich [Rich et al., 91] la llama *Mutually-Disjoint-With*. Dicha relación garantiza que entre una o más clases no existe ningún elemento común.

Dos marcos pueden relacionarse mediante la **Relación No Disjunta**, en inglés *Non-Disjoint Relationship* [Frost, 86], si representan conjuntos no disjuntos de entidades.

Dos marcos se pueden relacionar mediante la **Relación Similar** [Frost, 86], en inglés *Similar Relationship*, si representan conjuntos de entidades cuyos miembros tienen propiedades en común suficientes para ser considerados como similares. El significado de "similar" es subjetivo y dependerá del empleo que se haga del sistema de Marcos.

La **Relación Recubrimiento**, en inglés *Is-Covered-By* [Rich et al., 91], relaciona una clase con el conjunto de sus subclases, de tal forma que la unión de dichas subclases es la clase inicial. Si una clase está cubierta por un conjunto de clases mútuamente disjuntas, entonces, se dice que el conjunto de las clases es una partición de la clase.

La relación **Compuesto de** o **Forma parte de** permite describir las relaciones existentes entre todos los componentes físicos que forman un sistema en estudio, formando así una jerarquía que representa, para cada elemento del sistema, los componentes que lo forman.

Los Marcos pueden relacionarse mediante relaciones **complejas** [Frost, 86] como en el ejemplo que se ilustra en la Figura 2.4

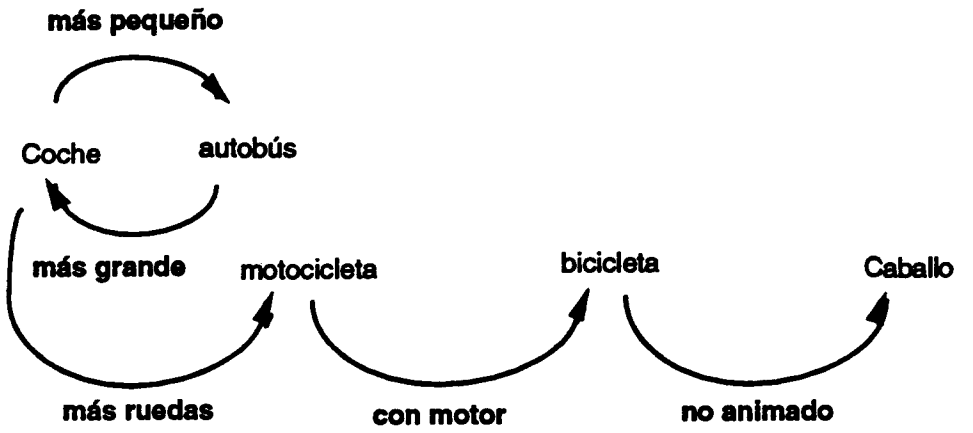


Figura 2.4. Relaciones complejas entre conjuntos de entidades.

Todas estas relaciones, y aquellas que el usuario pudiera definir "ad hoc", en una jerarquía se expresan definiendo en el marco origen de la relación una ranura que tiene como nombre el de la relación y como valor el destino de la relación.

Una clasificación de las relaciones que aparecen en un SBM es la mostrada por Gómez [Gómez, 93] que distingue dos tipos de relaciones, según permita aplicar o no herencia de propiedades, relaciones que reciben el nombre de **Relaciones Estándares** y **Relaciones No Estándares** respectivamente. Las primeras incluyen a las **Relaciones SubClase** e **Instancia**, y las segundas, a las **Relaciones Fraternal, Disjunto, No Disjunto, Similar, Compuesto de** o **Forma Parte de** y todas las relaciones "ad hoc" que defina el usuario.

2.2.2.2 Descripción de los Atributos: Propiedades y Facetas

Dado que una clase o conjunto se representa en la Teoría de Marcos utilizando marcos clase, las propiedades o atributos de dichas clases se clasifican en tres categorías [Nado et al., 92]:

a) Propiedades que dan nombre a la clase.

b) Propiedades que describen la clase como una entidad.

c) Propiedades que describen prototipos.

En cualquier caso, las propiedades, al igual que ocurría con las relaciones, se van a representar en el Marco Clase mediante un conjunto de ranuras. Básicamente, la

mayoría de los autores distinguen dos tipos de propiedades: propiedades que describen características genéricas de una clase y propiedades que se utilizan para describir cada miembro de la clase. A estas propiedades, Fikes y Kehler [Fikes et al., 85] las llaman *Own Slots* y *Member Slots*, que se traducirán al castellano como Propiedades de Clase y Propiedades de Instancia respectivamente.

a) Las **Propiedades de Clase**, describen atributos o características genéricas de un concepto o clase. Estas propiedades toman siempre el mismo valor en todos los elementos de la clase.

b) Las **Propiedades de Instancia**, describen propiedades comunes a todos los elementos del concepto o clase que el marco representa, pero son propiedades que se rellenan en la instancia con un valor u otro dependiendo del elemento concreto de la clase que se esté representando. Recientemente, Nado y Fikes [Nado et al., 92], han llamado a estas ranuras *Prototype Slots* por ser ranuras que se definen en la clase, que representan prototipos y que toman valor en las instancias de la clase.

En el *Standards and Review Manual for Certification in Knowledge Engineering* [Steinheiser, 90], además de las ranuras que representan las propiedades o atributos, se distingue otras ranuras que se utilizan para representar relaciones entre marcos. En estos casos, el nombre de la ranura será el nombre de la relación y el valor de dicha ranura es el nombre o los nombres de los marcos con los que este marco se encuentra unido.

Independientemente de que una ranura represente una propiedad de clase o de instancia o una relación, la ranura se va a rellenar con unos valores simbólicos o numéricos. Reichgelt [Reichgelt, 91] clasifica los valores que puede tomar una ranura en los siguientes grupos: ranuras que almacenan punteros a otros marcos, ranuras tipo par atributo-valor y ranuras que almacenan procedimientos.

a) **Ranuras que almacenan punteros a otros marcos.**

Las ranuras que contienen punteros a otros marcos pueden representar relaciones no estándares entre marcos o bien propiedades del marco que se definen utilizando a su vez marcos.

a.1) Si la ranura se utiliza para materializar relaciones entre marcos, la ranura se definirá siempre en el marco origen de la relación y se rellenará con un puntero al marco destino.

a.2) Si la ranura representa una propiedad de un marco que se define utilizando otros marcos de la BC, la ranura se rellena con un puntero al marco que define el tipo de valores correctos para esa propiedad. Este mecanismo, descrito por Block y Chan [Block et al., 89], permite aplicar herencia a nivel de ranura, al heredar la ranura la descripción realizada en el marco que la define. Esta herencia, herencia complementaria a la herencia de marcos, permite, además de compartir definiciones y valores de ranura en ranuras similares definidas en el mismo o en diferentes marcos, disminuir el número de marcos del SBM. Por ejemplo, supóngase el marco clase *Fecha* que describe el concepto Fecha. Supóngase también el marco clase *Persona* en el que se han definido las ranuras *Fecha_Nacimiento*, *Fecha_Fin_Carrera* y *Fecha_Boda*. La definición de estas tres ranuras mediante el marco clase *Fecha*, evita repetir la descripción del concepto Fecha en ellas. Se dice entonces que las tres ranuras heredan la descripción realizada en el marco que la define.

b) Ranuras tipo par atributo-valor.

Las ranuras tipo par atributo-valor asignan valores a un atributo de un marco. Existen varias formas de asignar valores a las ranuras:

b.1) Asignar un valor simple o multivaluado a una propiedad [Reichgelt, 91], [Rich et al., 91].

b.2) Representar explícitamente que el valor de la propiedad se rellena utilizando valores por omisión. Los valores por omisión son valores con los que se rellenan las propiedades de un marco instanciado si no se conoce explícitamente otro valor [Frost, 86]. Por ejemplo, la mayoría de los ratones tienen cuatro patas, a excepción de Mickey Mouse que tiene dos. El uso de este conocimiento está justificado por la razón siguiente: si hay conocimiento de lo contrario y éste es significativo, entonces es

probable que se de ese conocimiento. Así, si el número de patas que tiene un ratón es relevante para la aplicación, entonces se esperará que la BC contenga conocimento de ratones que tienen tres, dos, una o cero patas. En este caso, si no se dispone de este tipo de conocimiento, entonces, el valor cuatro sería correcto en la mayoría de los casos. Sin embargo, si el número de patas de un ratón es irrelevante en la aplicación, entonces, la BC no almacenará conocimiento de ratones que tengan tres, dos, una o cero patas. Entonces, el valor por defecto cuatro, puede ser incorrecto en múltiples casos, pero, dado que para la aplicación el número de patas es irrelevante, las suposiciones por omisión realizadas incorrectamente no son importantes.

En cualquier caso, marcos clase o marcos instanciados pueden anular el valor asociado por omisión a una propiedad de un marco clase situado en un nivel superior de la jerarquía [Reichgelt, 91], [Rich et al., 91].

b.3) No siempre es conocido el valor de una propiedad, pero sí se puede tener información sobre el conjunto de posibles valores con los que se puede rellenar la propiedad. La mayoría de los lenguajes de representación del conocimiento basados en Marcos permiten asociar restricciones o condiciones a una propiedad utilizando:

. - Conectivas lógicas: AND, OR y NOT,

. - Funciones o predicados, y

. - Valores definidos en otras propiedades de otro marco clase.

Las restricciones se pueden aplicar recursivamente y ayudan a preservar la integridad semántica de una BC al restringir el número y rango de valores permitidos para un atributo [Fikes et al., 85].

c) Ranuras que almacenan procedimientos

Las ranuras también pueden rellenarse utilizando un conjunto de procedimientos que se ejecutan cuando se requiere el valor de una ranura y éste no ha sido explícitamente almacenado. En KRL [Bobrow et al., 77] se distinguen dos tipos de procedimientos: sirvientes y demonios. Los sirvientes, en inglés *Servant*, se ejecutan cuando se necesita un valor de una ranura. Los demonios, en inglés *Demons*, se ejecutan cuando se actualiza o se añade un valor en una ranura.

Por tanto, dependiendo del tipo de la ranura así serán los valores que éstas tomarán, pudiéndose definir un conjunto de atributos que, por defecto, existirán en cada una de ellas. Al conjunto de posibles valores que puede tomar cada una de las ranuras que se hayan definido en cada marco clase de la jerarquía se le llama **Facetas**, en inglés *Facets*. En palabras de Fikes y Kehler [Fikes et al., 85], las facetas permiten incluir en el marco clase descripciones parciales de los valores de sus atributos y ayudan a preservar la integridad semántica de un SBC al restringir el número de posibles valores que un atributo puede tener y para indicar la clase a la cual el valor tiene que pertenecer. A continuación se exponen las facetas más utilizadas [Block et al., 89], [González et al., 93], [Rich et al., 91] y [Winston, 92].

Nexo
En esta faceta se definen todas aquellas ranuras que representan relaciones estándares y no estándares entre marcos. El valor con el que se rellena la ranura es el puntero al marco destino de la relación. El resto de ranuras no toman ningún valor en esta faceta.

Tipo
En esta faceta se especifica el nombre del marco que define el tipo de la ranura.

Valor Permitido
Se especifica en esta faceta el rango de valores entre los cuales el valor de la propiedad es válido.

Valor por Omisión
Se incluyen en esta faceta los valores con los que puede rellenarse las propiedades de un marco instanciado si no se conoce de forma explícita otro valor.

Condición

Los valores que toma una propiedad en un marco instanciado se restringen con condiciones que se definen en esta faceta del marco clase.

Cardinalidad Mínima

Una cardinalidad mínima "m" indica que, como mínimo, esa ranura se podrá rellenar con "m" valores diferentes.

Cardinalidad Máxima

Una cardinalidad máxima de "n" indica que, como máximo, esa ranura se podrá rellenar con "n" valores diferentes.

Definición

Se comenta en esta faceta la semántica de la ranura.

Si se Necesita

En la faceta "si se necesita" de cada propiedad de cada marco clase se definirá un procedimiento que se ejecutará cuando en un marco instanciado se desconozca el valor de la propiedad en cuestión o cuando se deseen calcular nuevos valores que ya están implícitos en la BC.

Si se Añade

En la faceta "si se añade" de cada propiedad de cada marco clase se definen procedimientos que se ejecutan cuando se añaden datos a una instancia de dicho marco clase.

Si se Modifica

En la faceta "si se modifica" de cada propiedad de cada marco clase se definen procedimientos que se ejecutan cuando en alguna instancia de dicho marco clase se modifiquen el/los valor(es) que toma la propiedad.

Si se Borra

En la faceta "si se borra" de cada propiedad de cada marco clase se definen procedimientos que se ejecutan cuando en alguna instancia de dicho marco clase se borren el (los) valor(es) que toma la propiedad.

Propiedad General

En esta faceta se definen las propiedades de clase asignadas a cada marco clase.

2.2.3 INFERENCIA EN MARCOS

El formalismo de Marcos proporciona un conjunto de técnicas que permiten realizar importantes y poderosas inferencias basándose principalmente en la taxonomía de los marcos y en sus propiedades. No obstante, la inclusión de reglas de producción en una BC basada en marcos proporciona un conjunto de inferencias adicionales que incrementan sustancialmente las inferencias que se pueden realizar en dicha jerarquía. Un SBM, a la hora de realizar inferencias, podrá utilizar, de forma integrada, algunas de las siguientes técnicas:

Equiparación [Frost, 86], [Reichegelt, 91]. Consiste en determinar qué marco de la jeraquía define mejor la situación actual.

Herencia [Reichgelt, 91]. Permite que los marcos de niveles inferiores de la jerarquía hereden las propiedades y los métodos definidos en los marcos de los niveles superiores.

Valores Activos y Métodos [Barr et al., 82], [Fikes et al., 85]. El conocimiento procedimental en marcos, expresado mediante procedimientos y reglas de producción, se utiliza para realizar inferencias en la BC. Los **Valores Activos** y **Métodos**, son las técnicas que permiten utilizar el conocimiento procedimental almacenado con el conocimiento declarativo en la jerarquía.

Con estas tres técnicas, el motor de inferencias tiene que ser capaz de:

* Clasificar entidades en la jerarquía [Steinheiser, 90]
* Inferir información adicional de una entidad concreta no declarada explícitamente en la jerarquía [Steinheiser, 90].
* Crear, modificar y borrar marcos y relacionarlos en ejecución [Walters et al., 88].

2.2.3.1 Equiparación

La **Equiparación** [Reichgelt, 91], en inglés *Matching*, consiste en determinar cuál es el marco clase que mejor describe la situación actual representada en la instancia. Dada la descripción de una situación, el proceso de equiparación tiene como

objetivo descubrir los marcos clase que se pueden aplicar, de forma consistente, a la situación actual.

Aunque los SBM no son los únicos que utilizan equiparación, *sí* puede decirse que el mecanismo de equiparación en marcos es mucho más complicado que en el resto de los formalismos de representación del conocimiento. Reichgelt [Reichgelt, 91] aporta los argumentos siguientes a favor de esta hipótesis:

1. La estructura que se va a equiparar es más complicada que la estructura que, por ejemplo, presentan las reglas de producción.

2. Una segunda razón se deriva de las propiedades por omisión que se han asociado a los marcos clase, pues estos marcos no siempre representan información cierta para todas las instancias. Por ejemplo, aunque todos los *perros* tienen la lengua de color rojo, la raza *chau-chau* la tienen de color azul.

3. En ocasiones, se pueden realizar varias equiparaciones correctas de un marco instanciado con varios marcos clase.

En muchas aplicaciones de los marcos, el tipo de una entidad no siempre es conocido [Frost, 86]. Por ejemplo, dada una entidad que representa un *Animal*, se podría conocer que es un *Mamífero* y que tiene *Cuatro Patas*, pero ignorar su *Raza* y su *Especie*. En tales casos, no siempre es sencillo seleccionar un marco e instanciarlo después para representar la entidad entre manos. Para ello, hay que utilizar las propiedades conocidas para seleccionar un marco candidato. El sistema trata entonces de encontrar valores para rellenar las propiedades que no lo estén, dándose las dos situaciones siguientes [Frost, 86]:

1. Si se encuentran valores no apropiados, entonces se debe seleccionar otro marco candidato. La identificación de marcos candidatos apropiados, puede ser facilitada en algunos casos, mediante las referencias a otros marcos, teniendo así nuevos marcos a explorar [Rich, 83].

2. Si no se encuentran valores, entonces el sistema puede seguir suponiendo que el conocimiento ha desaparecido.

En efecto, el sistema trata de encontrar el marco que se equipare mejor con las propiedades conocidas de una entidad. Para llevar a cabo este emparejamiento, se puede

usar el siguiente proceso: Supóngase que se conocen los valores de varias propiedades de la entidad E. Entonces, el equiparador del sistema de marcos, puede utilizar esas propiedades para seleccionar el marco candidato. Esto puede realizarse de distintas formas:

a) Indicar los marcos mediante los nombres de las ranuras, seleccionando cualquier marco que tenga un nombre de ranura que sea el mismo que cualquiera de los nombres de propiedad para los cuales hay un valor conocido para E. Se elige el marco que mejor equipara con la situación planteada [Rich, 83], [Frost, 86].

b) Utilizar información contextual para seleccionar un marco [Frost, 86]. Por ejemplo, si se sabe que la entidad E es un perro, entonces, seleccionar el marco *Perro* y no el marco *Mamífero* o el marco *Gato*.

c) Comenzar en la parte superior de la jerarquía y seleccionar el marco con el mejor Valor de Equiparación en el primer nivel [Frost, 86].

d) Recorrer la jerarquía, de abajo hacia arriba, hasta encontrar un marco que sea lo suficientemente general que no contradiga la instancia [Rich, 83].

e) Seleccionar un marco arbitrariamente.

Cuando se ha seleccionado un marco candidato utilizando uno de los métodos anteriores, el sistema instancia parcialmente el marco, rellenando tantas propiedades como sea posible. En algunas aplicaciones, el sistema puede pedir conocimiento adicional para intentar rellenar más propiedades.

A continuación, se calcula el **Valor de la Equiparación**, en adelante VE, [Frost, 86] que indica el grado de idoneidad de la equiparación realizada. Por ejemplo, si todos las propiedades se rellenaran con valores apropiados, entonces el VE sería 1. Si se rellenase sólo la mitad de esas propiedades, el VE sería de 0,5 para el mismo peso de cada propiedad.

El método de cálculo del VE, variará de unas aplicaciones a otras. En algún caso, la presencia de un valor de una propiedad inadecuada, fijará un VE igual a cero. Por ejemplo, una entidad con valor de la ranura *Sangre* igual a *Fria*, tendría un VE cero cuando se la emparejara con el marco *Persona*. En otros casos, los valores de las

ranuras no apropiadas podrían no ser tan esenciales y se limitarian sencillamente a reducir el VE.

Dependiendo de la aplicación, se considerará que el VE significa varios grados de éxito. Si el VE es lo suficientemente alto, y el marco es lo suficientemente específico para la aplicación que se está haciendo, entonces el sistema no buscará otros marcos. Sin embargo, si el VE no es lo suficientemente alto, o el marco no es lo suficientemente específico, entonces el sistema considerará otros marcos. En lugar de mirar el resto de marcos, el sistema puede hacer uso de la estructura o taxonomía en marcos para identificar marcos relevantes. Esto puede realizarlo de varias maneras, entre las que cabe destacar las siguientes [Frost, 86]:

a) Ascender a lo largo de la jerarquía por relaciones de tipo SubClase hasta encontrar una equiparación perfecta. Podría entonces buscarse en el subárbol situado por debajo del marco equiparado para intentar encontrar una equiparación más específica. Esa búsqueda, dependiendo de los casos, podría ser ciega o heurística. Por ejemplo, si se equipara una entidad E con un marco *Mono* y se obtiene un VE de 0,3, se podría ascender por la jerarquía utilizando la relación *SubClase* y encontrar una equiparación perfecta con, póngase por caso, el marco *Mamifero* para intentar equiparar E con un marco más específico tal como un marco *Perro*.

b) Se podrían buscar marcos relacionados con el que se tiene entre manos mediante las relaciones: Similar o Fraternal. Por ejemplo, si una entidad E equiparaba el marco *Mujer* con un VE de 0,9, entonces el sistema podría intentar equiparar E con marcos tales como *Hombre* para intentar encontrar una equiparación mejor.

c) Utilizar relaciones complejas entre marcos, como las que aparecen en la figura 2.4, para identificar marcos candidatos a la equiparación. Por ejemplo, si una entidad equipara el marco *Coche* excepto para su *Número de Ruedas*, que es *dos*, entonces la relación *Similar Pero Menos Ruedas*, podría utilizarse para identificar el marco *Motocicleta* como candidato a una equiparación idónea.

Los Sistemas de marcos se usan a menudo como sistemas de reconocimiento de patrones, tanto si los patrones son visuales como si representan conceptos abstractos

[Frost, 86]. Por ejemplo, un sistema de marcos podría constar de un conjunto de marcos, uno para cada letra del alfabeto. El marco para la letra *E* podría contener ranuras y los valores tales como: *Número de líneas rectas = 4*, *Número de ángulos = 2* y *Número de líneas curvas = 0*. Tal sistema podría emplearse, entonces, en el reconocimiento de letras mayúsculas. En este caso, el problema que ha de resolverse es meramente un problema de reconocimiento de patrones. En un sistema de marcos, el reconocimiento de patrones implica la "equiparación" de un conjunto de valores asociados a una entidad con los valores requeridos para rellenar las ranuras de un marco. Si todas las ranuras de un marco M pueden rellenarse con valores apropiados que se relacionen con una entidad E, entonces M puede instanciarse para representar E y E puede clasificarse como tipo M. Sin embargo, M podría representar un conjunto entidad "general" tal como *Persona*. Si se quiere encontrar una equiparación más específica, debe examinarse los niveles inferiores de la jerarquía para ver si una especialización de M puede ser instanciada. Por ejemplo, se podría encontrar que los valores relacionados con E fueran apropiados para rellenar las ranuras de un marco N que representa *Informático*. Equiparar E con N es más informativo que emparejar E con M.

En otras aplicaciones, cuando, verbigracia, se ha reconocido que una entidad es un tipo de entidad particular, todas las propiedades genéricas asociadas con el tipo se heredan por esa entidad. El sistema actúa entonces en la modalidad de reconocimiento de patrones e inferencia.

2.2.3.2 Herencia de Propiedades

Una vez que un marco instanciado se ha equiparado con un marco clase, el sistema que realiza la inferencia, utilizando la información almacenada en la BC, comienza a generar expectativas sobre él. El mecanismo que permite pasar o aplicar información general a la situación actual concreta recibe el nombre de **Herencia**, en inglés *Inheritance*, y es una característica intrínseca a la Jerarquía de Marcos. La Herencia permite compartir información de propiedades y métodos, y aplicar dicha información a la situación actual.

Patel-Schneider [Patel-Schneider, 91] distingue dos tipos de herencia. La **Herencia basada en Nociones Procedimentales** en la que se define el algoritmo de herencia en función de las estructuras de los datos con los que trabaja. Estos sistemas carecen de semántica, son poco expresivos y tienen un limitado poder

deductivo. El segundo tipo de herencia, llamado **Herencia no Procedimental**, tiene asociado modelos teóricos formales que proporcionan la semántica a los sistemas de herencia en grafos acíclicos dirigidos. El precursor de este tipo de herencia es Touretzky, que desarrolló una Teoría Matemática Formal de Herencia múltiple, no monótona, homogénea y bipolar en grafos acíclicos dirigidos en su Tesis Doctoral [Touretzky, 84]. Dos años más tarde, en 1986, Thomason, Horty y Touretzky [Thomason et al., 86] desarrollaron otra Teoría Matemática de Herencia múltiple, monótona y bipolar en redes semánticas. En 1987, los mismos autores, crearon una Teoría Formal de Razonamiento escéptico no monótono en redes semánticas [Horty et al., 87]. Desde entonces, muchos investigadores han estudiado la herencia de propiedades en grafos acíclicos dirigidos utilizando este enfoque formal. Prueba de ello son los artículos presentados en el *Workshop on Inheritance Hierarchies in Knowledge Representation and Programming Languages* [Lenzerini et al., 91].

2.2.3.2.1 Terminología Básica

Antes de pasar a analizar la terminología básica de los sistemas de herencia se va a explicar la nomenclatura utilizada, propuesta por Touretzky [Touretzky, 84] en su Teoría Formal. Si a es un individuo y p y q son clases, el arco $a \rightarrow p$ y $a \not\rightarrow q$ equivale a decir en lógica $p(a)$ y $\neg\, q(a)$. Los arcos entre clases, como $p \rightarrow q$ y $p \not\rightarrow q$ no tienen una interpretación lógica y, dependiendo del sistema de herencia en el que aparezcan, pueden representar sentencias del Cálculo de Predicados de Primer Orden, Lógica de Defectos o Lógica no Monótona.

Touretzky, Horty, Thomason [Touretzky et al., 87] clasifican a los distintos tipos de sistemas de herencia en:

a) Sistemas de herencia simple o múltiple

b) Sistemas de herencia bipolares o polares

c) Sistemas de herencia no monótono o monótono

d) Sistemas de herencia homogéneos o heterogéneos.

a) Sistemas de Herencia Simple o Múltiples

Un sistema de herencia que se aplica a una red en forma de árbol, recibe el nombre de sistema de herencia simple. En estos sistemas, sólo existe un único camino

que une el nodo actual con el nodo raíz de la jerarquía. Por el contrario, un sistema de herencia múltiple, no restringe el número de nodos con los que el nodo actual puede estar unido. En estos sistemas la red es un grafo acíclico dirigido.

b) *Sistemas de Herencia Bipolares o Polares*

Los sistemas de herencia bipolares contienen enlaces positivos (→) y negativos (↛) mientras que los polares solamente contienen enlaces positivos. Por ejemplo, en un sistema bipolar se podría representar que *Mickey Mouse es un Ratón de Walt Disney* y que *Mickey Mouse no es un Perro de Walt Disney* como: *Mickey Mouse→Ratón de Walt Disney* y como *Mickey Mouse ↛ Perro de Walt Disney*. En un sistema polar la relación negativa se representaría utilizando una relación positiva hacia un nodo *No Perro de Walt Disney*, es decir, *Mickey Mouse→No Perro de Walt Disney*.

c) *Sistemas de Herencia no Monótono y Monótono*

Los sistemas de herencia no monótonos trabajan con excepciones al aplicar la herencia de propiedades mientras que los monótonos no. Touretzky [Touretzky, 86] describe un sistema no monótono bipolar que trabaja con las excepciones de forma implícita. Otros sistemas, como son la Lógica no Monótona [McDermott et al., 80] y la Lógica de Defectos [Reiter, 79], [Etherington et al., 83], trabajan con excepciones de manera explícita. Nado y Fikes [Nado el al., 87] construyeron un lenguaje formal de Marcos basado en el formalismo de Etherington [Etherington, 87] que utilizaba Sistemas de Mantenimiento de la Verdad [Doyle, 79] para trabajar con las excepciones.

d) *Sistemas de Herencia Homogéneos y no Homogéneos o Heterogéneos*

Los sistemas de herencia homogéneos son, en todas las ocasiones, monótonos o, en todas las ocasiones, no monótonos, mientras que los no homogéneos en ocasiones son monótonos y en otras no monótonos.

Independientemente de que la herencia sea simple o múltiple, polar o bipolar, homogénea o heterogénea, monótona o no, un marco instanciado puede heredar para una propiedad concreta uno o varios valores definidos en la jerarquía, hablándose entonces de **Herencia no Multivaluada** y de **Herencia Multivaluada**, en inglés *Single-Valued Inheritance* y *Multivalued Inheritance* respectivamente. A continuación, se describen las técnicas de búsqueda asociadas a la herencia simple y múltiple no multivaluada en sistemas de herencia procedimentales y no procedimentales.

2.2.3.2.2 Sistemas de Herencia Procedimentales

En este apartado se exponen, desde un punto de vista procedimental, los métodos de herencia simple y múltiple más utilizados en el formalismo de Marcos y los problemas que cada uno plantea.

a) Herencia Simple

Se aplica la herencia simple a una jerarquía de marcos cuando sólamente existe un único camino que une el marco instanciado con el nodo raiz de la jerarquía, es decir, cuando cada marco instanciado es instancia de un único marco clase y cuando cada marco clase es, a su vez, especialización de una única clase. Entonces, el tipo de herencia que se realiza sobre el árbol o taxonomía de marcos recibe el nombre de herencia simple. La técnica de búsqueda que se utiliza para recorrer el árbol es una especie de búsqueda en profundidad desde el nodo instanciado hacia el marco clase raíz de la jerarquía.

El algoritmo que realiza la inferencia es el siguiente:

Paso1: Buscar la propiedad en el marco instanciado.

a) Si se encuentra la propiedad, el sistema devuelve como valor de la propiedad el/los valores almacenados en la ranura.

b) Si no la encuentra, utilizando la relación de instanciación, va al marco padre.

Paso 2: Buscar la propiedad en el marco padre.

a) Si encuentra la propiedad, devuelve como valor de la propiedad el/los valores almacenados en la ranura.

b) Si no la encuentra, utilizando la relación SubClase, va a su padre. Se repite el paso 2 si el padre no es el nodo raíz de la jerarquía.

Paso 3: Buscar la propiedad en el nodo raíz de la jerarquía.

a) Si la encuentra, se devuelve el valor almacenado.

b) Si no, el sistema responde que, con el conocimiento almacenado en la jerarquía, es imposible responder a la pregunta planteada.

Al aplicar este algoritmo, si en el camino que une el marco instanciado con el marco raíz de la jerarquía se encuentran en marcos distintos la misma propiedad con diferentes valores, se hereda siempre el valor de la propiedad situada en el marco de nivel inferior, es decir, el conocimiento más específico prevalece sobre el conocimiento más general, al ser el orden en el que se recorre el grafo un orden total.

b) Herencia Múltiple

Se aplica la herencia múltiple [Touretzky et al., 87] a una jerarquía de marcos cuando existen varios caminos que unen el marco instanciado con el nodo raíz de la jerarquía, es decir, cuando cada marco instanciado puede ser instancia de varios marcos clase y, o, cuando cada marco clase es especialización de una o varias clases. Entonces, el tipo de herencia que se realiza sobre el grafo dirigido acíclico que representa la jerarquía de marcos recibe el nombre de herencia múltiple. Fahlman [Fahlman, 79] llamó a estos grafos *Tangled Hierarchies,* jerarquías que requieren nuevos algoritmos para realizar inferencias. En este tipo de jerarquías, un hijo puede heredar propiedades y métodos de varios padres y la dificultad se encuentra en determinar el orden en el cual las clases deben ser exploradas. Si los padres tienen propiedades y métodos con distinto nombre nunca habrá conflicto a la hora de heredar una propiedad o método y su valor. Sin embargo, si los padres tienen propiedades y, o, métodos con el mismo nombre, entonces, el valor en el marco hijo dependerá de la técnica de búsqueda que se haya empleado para recorrer el grafo. Las técnicas de búsqueda empleadas para recorrer el grafo son una especie de búsqueda en amplitud y en profundidad desde el marco instanciado hacia el marco clase raíz. A continuación, se describen cada una de ellas.

b.1) Búsqueda en profundidad

Esta técnica explora en profundidad todos los posibles caminos que van desde el marco instanciado al marco clase raíz de la jerarquía. Los criterios que deben estar presentes a la hora de recorrer en profundidad la jerarquía son [Winston, 92]:

1. Dado que una instancia puede pertenecer a varias clases y que una clase puede ser subclase de varias clases, es necesario establecer un criterio adicional que evite el conflicto, por ejemplo, el criterio estándar de buscar de izquierda a derecha.

2. Con el fin de evitar la búsqueda reiterada de propiedades en marcos clases ya analizados se añadirá, como criterio adicional, el criterio de exhaustividad, es decir, sólamente se buscará la propiedad en cada marco clase de la jerarquía una única vez.

3. El problema que presenta la búsqueda exhaustiva en profundidad de izquierda a derecha es que las instancias pueden heredar, para una propiedad que presenta excepciones, primero los valores de las clases más generales, en vez de heredar los valores de marcos clases que son especializaciones de los primeros. Esta situación ocurre cuando el marco que presenta la excepción está situado en la parte derecha de la jerarquía. Con el fin de paliar este error, se introduce el siguiente criterio en el procedimiento de búsqueda: *Cada clase debe aparecer en la lista de clases de precedencias antes que todas sus superclases.*

Un algoritmo que cumple los anteriores requisitos es el procedimiento de ordenación topológica [Bobrow et al., 90]. Este procedimiento, en función de la topología de los nodos de la jerarquía, determina el orden en el que se van a heredar propiedades. Básicamente, el algoritmo es el siguiente:

Paso1: Crear una *lista de nodos* recorriendo todos los caminos del árbol que van desde el nodo instanciado hacia el nodo raíz. La forma de recorrer el árbol es búsqueda en profundidad, exhaustiva y de izquierda a derecha.

Paso 2: Crear, para cada nodo que aparezca en la lista anterior, su *lista de pares de elementos*. Por ejemplo, la lista de pares de elementos asociada a la figura 2.5 es: (A, B), (B, C) y (C, D).

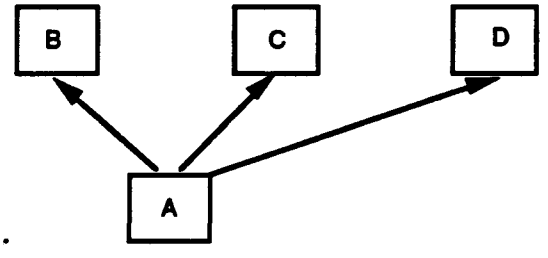


Figura 2.5. Ejemplo de jerarquía

Paso 3: Determinar, para el conjunto de pares de elementos obtenidos en el paso 2, un átomo que aparezca como primer elemento de una o varias listas y nunca como segundo elemento. Una vez localizado el átomo, se borran los pares de elementos en los que el átomo aparece como primer elemento y

se incluye este átomo dentro de la lista de criterios de preferencia. El proceso continúa mientras haya elementos en la lista de pares de elementos.

b.2) Búsqueda en amplitud

Esta técnica consiste en explorar en amplitud todos los posibles caminos que van desde el marco instanciado al marco clase raíz del sistema de Marcos. Dado un marco instanciado, se miraría si la propiedad buscada se encuentra en alguno de los marcos clases del nivel superior con los que dicho marco instanciado está relacionado. Si se encuentra, se devuelve el valor almacenado en la ranura, si no, se asciende, utilizando la relación SubClase de cada marco clase, un nivel en la jerarquía y se comienza a buscar en todas las superclases de las clases anteriomente mencionadas. El proceso termina cuando se encuentra la ranura buscada o cuando se alcanza el nodo raíz de la jerarquía. En este último caso, y si no encuentra la propiedad buscada en el nodo raíz, el sistema devuelve que desconoce el valor asociado a la propiedad por no encontrarse ésta almacenada en la jerarquía.

Este algoritmo utiliza la longitud del camino para realizar inferencias. Algunos lenguajes que utilizan la longitud del camino para realizar herencia son FRL [Roberts et al., 77] y NETL [Fahlman, 79]. Básicamente, el algoritmo consiste en lo siguiente: partiendo del marco instanciado se busca en amplitud la propiedad en la jerarquía. Si por un camino encuentra una solución, entonces devuelve el valor y termina, pues si para la longitud actual los caminos restantes no han encontrado ningún valor de la propiedad, sus longitudes serán siempre superiores a la del camino de éxito. El algoritmo que utiliza la longitud del camino es eficiente y es correcto en sistemas de herencia que razonan sobre taxonomías y sobre jerarquías que no admiten excepciones, es decir, sistemas monótonos. Sin embargo, cuando la herencia múltiple se combina con excepciones (sistemas de herencia no monótonos) las inferencias que se realizan son mucho más complejas y presentan dos problemas semánticos [Touretzky, 84]: razonar en presencia de redundancias y razonar en presencia de ambigüedades.

Un ejemplo que muestra el razonamiento en sistemas de herencia múltiple no monótonos en presencia de redundancias es el de la figura 2.6. La redundacia introducida por el enlace positivo desde *Clyde* hacia *Elefante* ocasiona problemas en la inferencia porque existen dos caminos de igual longitud que son independientes y que llevan a heredar para una misma propiedad, valores contradictorios. La solución que Touretzky [Touretzky, 84] propone no es eliminar o prohibir el enlace desde *Clyde*

hacia *Elefante*, sino primar la información almacenada en *Elefante Real* sobre la información de *Elefante*.

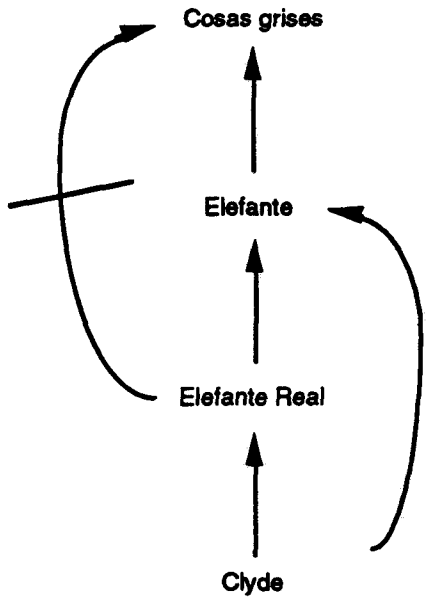


Figura 2.6. Ejemplo de Conocimiento Redundante

El segundo problema que puede aparecer en un sistema de herencia múltiple no monótono es la detección de ambigüedades, pues no todos los sistemas de inferencia son capaces de reconocerlas. Los algoritmos que se basan en la longitud del camino eligen una de las conclusiones arbitrariamente o responden que, simultáneamente, las dos conclusiones son ciertas. La incapacidad de reconocer ambigüedades es otro motivo por el que la longitud del camino es inaceptable para razonar en este tipo de situaciones. La figura 2.7 [Touretzky, 86], muestra el conocido ejemplo del *rombo de Nixon* en el que se razona en presencia de ambigüedades al concluir que *Nixon* es simultáneamente tanto un *pacifista* como un *no pacifista*.

A estos problemas Rich y Knight [Rich et al., 91] añaden que la longitud del camino en los Marcos, en ocasiones, no se corresponde con el nivel de generalidad de la clase. La longitud del camino depende de la elaboración o grano de conocimiento almacenado en la BC. Así, si el conocimiento está muy elaborado, la longitud del camino desde el marco instanciado a un marco clase es mayor que si no lo está. Esto ocasiona problemas cuando, para una clase, se ha desarrollado mucho una de sus subclases y muy poco otra. La figura 2.8 describe esta situación.

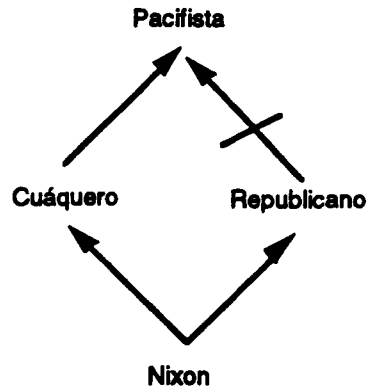


Figura 2.7. Ejemplo de Conocimiento Ambiguo

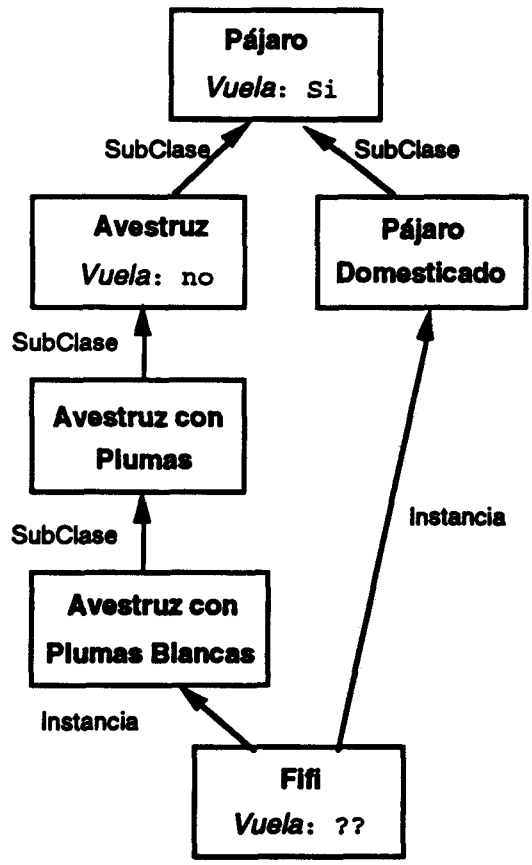


Figura 2.8. Ejemplo de Conocimiento no Especializado

2.2.3.2.3 Sistemas de Herencia No Procedimentales o Formales

Touretzky [Touretzky, 84] presenta una Teoría Matemática Formal de herencia en grafos acíclicos dirigidos basada en el concepto de distancia "inferencial". La distancia "inferencial" permite utilizar la herencia con sentencias redundantes ciertas en la BC y detectar, pero no resolver, situaciones ambiguas. El sistema de herencia propuesto por Touretzky es bipolar, no monótono, homogéneo y múltiple. Supóngase que se desea conocer el valor de la propiedad P en A. Supóngase que existe un camino entre A y B y que B es una subclase de C. La propiedad P se encuentra definida en B y ¬ P en C. La distancia "inferencial" dice que si A tiene un camino vía B hacia C y no viceversa, entonces, se puede concluir que A tiene P. Pero, si A tiene un camino vía C hacia B y no viceversa, entonces, se concluirá que A tiene ¬ P.

La definición formal de distancia "inferencial" es:

> *La condición necesaria y suficiente para que un camino de inferencia de la forma: A →...→B →P deba prevalecer sobre el camino A →...→C ↛ P es que exista un camino A →...→B→...→C que establezca que para A, B es una subclase de C.*

Por tanto, en el ejemplo de la figura 2.6, *Clyde* está más cerca de *Elefante Real* que de *Elefante*, puesto que tiene un camino de inferencia a través de *Elefante Real* hacia *Elefante*, pero no viceversa. Un motor de inferencia que utilizara la distancia "inferencial" para realizar la herencia debería concluir que *Clyde* no es gris.

A diferencia del orden topológico que introduce un orden total, la distancia "inferencial" introduce un orden parcial [Touretzky, 84], pues en ésta última aparecen marcos conectados por enlaces de tipo clase. Por este motivo, al no permitir comparar marcos no conectados, el orden introducido por la distancia "inferencial" es un orden parcial y, por consiguiente, se pueden herededar valores contradictorios definidos en ramas que no están conectadas y tener ambigüedades. Al existir elementos no comparables, los sistemas de inferencia que utilizan la distancia "inferencial" no podrán seleccionar entre dos o más inferencias. En estos casos, el sistema no debería elegir o bien debería razonar sobre las consecuencias de cada una de sus elecciones. Este nuevo enfoque, independiente del anterior, es desarrollado por Horty, Thomason y

Touretzky [Horty et al., 87]. Ellos, introducen dos nuevos conceptos: el razonador escéptico y el razonador crédulo. El **razonador escéptico** es aquél que evita realizar conclusiones en situaciones ambiguas, mientras que el **razonador crédulo** intentaría concluir en la misma situación tanto como le fuera posible, aislando las conclusiones contradictorias de las que no lo son. Supóngase que se tiene un razonador escéptico que se aplica al ejemplo de la figura 2.7, el razonador no puede concluir que *Nixon* es un *pacifista* porque puede construir un argumento en contra. Igual sucedería si se quiere concluir que *Nixon* es un *no pacifista*. Básicamente, el razonamiento de los autores se basa en que un camino tiende a neutralizar a otro. No obstante, existen dos restricciones:

* *Caminos compuestos se pueden neutralizar en su totalidad.*
* Los caminos se neutralizan por caminos conflictivos más específicos.

a) Inconsistencias en Caminos Compuestos "versus" Caminos Directos

Supóngase que se tiene la red de la figura 2.9. En este caso, si *A* representa a *Nixon*, *p* a los *pacifistas* y $\neg p$ a los *no pacifistas*, el sistema concluiría que *Nixon* es tanto un *pacifista* como un *no pacifista*. Aunque aparentemente el caso de la figura 2.7 es similar a este último caso, existen importantes diferencias. En el primero, la información almacenada en la BC es consistente y existen dos caminos: $Nixon \rightarrow Cuáquero \rightarrow Pacifista$ y $Nixon \rightarrow Republicano \not\rightarrow Pacifista$, que llevan al conflicto final. Sin embargo, en el segundo, la BC ya contiene la inconsistencia $A \rightarrow P$ y $A \not\rightarrow P$. Horty, Thomason y Touretzky resuelven el problema utilizando la propuesta de Belnap [Benalp, 77] que guía el razonamiento deductivo en presencia de inconsistencias. Como principio general, los autores proponen que el razonador debería ser capaz de concluir a partir de un conjunto de sentencias, cualquier sentencia que actualmente esté contenida en el conjunto, incluso si el conjunto es inconsistente. Touretzky y colegas [Touretzky et al., 87], analizan, además, las inferencias realizadas por razonadores escépticos y crédulos que razonan hacia abajo y hacia arriba en grafos acíclicos dirigidos.

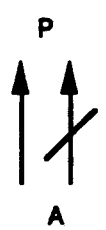


Figura 2.9. Inconsistencia en caminos directos: $A \rightarrow P$ y $A \not\rightarrow P$

b) Caminos Neutralizados

Supóngase el grafo de la figura 2.10. Si se preguntara si *Tweety* vuela, se llega a una contradicción originada en los caminos: *Tweety→Pingüino→Pájaro →Vuela* y *Tweety→Pingüino -/→ No Vuela.*

Dado que ambos caminos son compuestos y permiten conclusiones conflictivas, un razonador escéptico impediría alcanzar una conclusión. Sin embargo, a simple vista se observa que el razonador debería concluir que *Tweety no vuela* porque es un *pingüino* y los *pingüinos*, aunque son *pájaros*, *no vuelan.* Es decir, la información almacenada en *pingüino* debería anular la información de *pájaro*, o lo que es lo mismo, la información más específica debería anular la infomación menos específica.

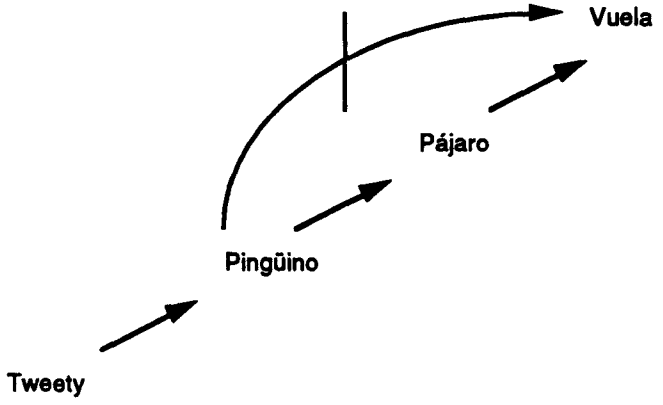


Figura 2.10. Jerarquía del conocimiento sobre *Tweety*

Por tanto, un camino compuesto es neutralizado por cualquier camino conflictivo que sea más específico que él. Formalmente, se define un camino compuesto [Horty et al., 87] como:

> *Supuesta una red, un camino de la forma x→r→v→y es anulado en una red si existe un nodo z en la red tal que z -/→ y, o bien z=x o la red permite un camino de la forma x→r1→z→r2→v.*

Gráficamente se muestran las dos situaciones en la figura 2.11.

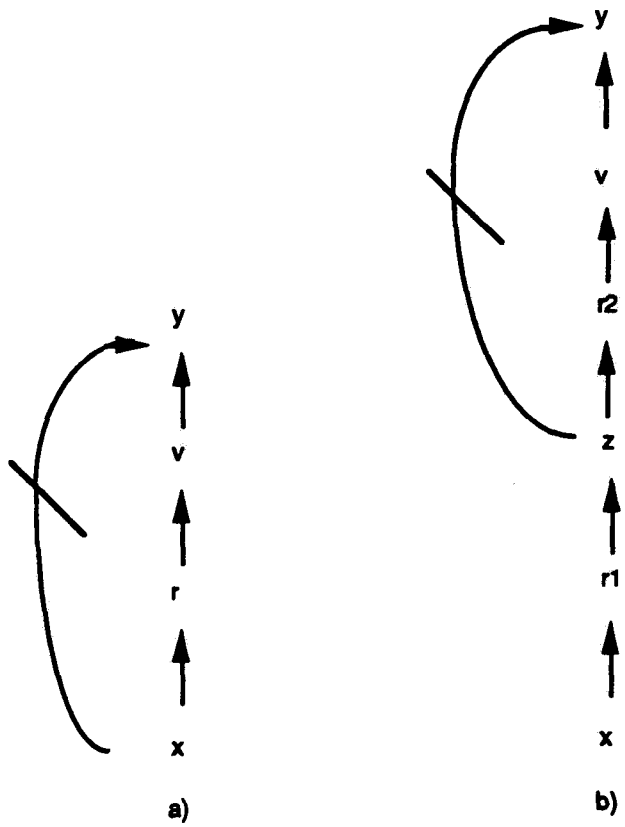


Figura 2.11. Caminos Neutralizados

En 1989, Horty y colegas [Horty et al., 89] combinan la Teoría de Touretzky [Touretzky, 86] y la de Horty, Thomason y Touretzky [Horty et al., 87] elaborando una Teoría del Razonamiento Crédulo que integraba los enfoques independientes. En el mismo año, Stein [Stein, 89] define la **Herencia Escéptica Ideal** como una forma de herencia que soporta exactamente aquellas inferencias ciertas en todas las extensiones crédulas de una jerarquía de herencia. Selman y Levesque [Selman et al., 89] muestran que la noción de herencia introducida por Touretzky es NP y que es necesario establecer un equilibrio entre la expresividad del formalismo y el esfuerzo computacional requerido para realizar las inferencias.

El algoritmo propuesto por E. Rich [Rich et al., 91] para realizar la búsqueda de la propiedad en un sistema de herencia de propiedades, múltiple, no monótono, suponiendo desconocido el valor que toma la ranura R en el marco M, es el siguiente:

Paso 1: Inicializar la lista de CANDIDATOS con vacía.

Paso 2: Realizar la búsqueda en amplitud o en profundidad partiendo del marco M siguiendo los enlaces estándares entre marcos. En cada paso, se comprueba si se ha encontrado en un marco un valor para la ranura R.

a) Si se encuentra el valor, se introduce el marco en la lista de CANDIDATOS y se termina de explorar esta rama.

b) Si no se encuentra, por los enlaces de tipo SubClase se sube un nivel más en la jerarquía.

c) Si no quedan más ramas sin explorar se va a 3.

Paso 3: Para cada elemento C de CANDIDATOS:

a) Comprobar si en la lista de CANDIDATOS se encuentra una clase más cercana a la instancia M que la clase C que se encuentre en el camino que une M con C.

b) Si existe, entonces se borra C de la lista de CANDIDATOS.

Paso 4: Analizar el número de elementos de la lista de CANDIDATOS

a) Si es cero, entonces, no se ha encontrado ningún valor para la ranura R.

b) Si es uno, entonces, se devuelve el valor que toma la ranura.

c) Si es mayor que uno, entonces, existe una contradicción o ambigüedad.

Stefik y Bobrow [Stefik et al., 86] muestran como algunos lenguajes de programación orientados a objetos y como algunas herramientas trabajan con la herencia de propiedades y métodos.

2.2.3.3 Conocimiento Procedimental

Subyacente y dependiente de la estructura declarativa de los marcos, es decir, de la manera en que se organizan los hechos estáticos, existe un aspecto dinámico o procedimental de los mismos. En muchos sistemas, los procedimientos asociados a cada marco son el mecanismo principal para dirigir la inferencia. Estos procedimientos, normalmente, están en estado latente en el marco clase, sin hacer nada, a menos que por algún motivo se solicite su ejecución. Su activación se produce cuando se requiere, se necesita, se modifica, se añade o se borra, entre otros, un dato almacenado en una

ranura de un marco instanciado. La finalidad de estos procedimientos es incorporar al sistema modelos de conducta de los objetos que intervienen y el conocimiento de expertos en el dominio [Fikes et al., 85]. A estos procedimientos normalmente se les llama, siguiendo la nomenclatura de la herramienta KEE [Fikes et al., 85], Valores Activos o Demonios y Métodos. Estos dos tipos de procedimientos se diferencian por la forma de su activación. Si la activación del procedimiento se realiza mediante una señal asíncrona, que llega a la BC desde el exterior, se está en presencia de los **Métodos**; si, por el contrario, el procedimiento se ha activado asíncronamente, en función de los datos almacenados en la BC, se está en presencia de **Valores Activos** o **Demonios.**

2.2.3.3.1 Demonios o Valores Activos o Disparadores

Con estos tres nombres, en inglés *Demons, Active Values y Triggers,* se conocen a los procedimientos, definidos en una ranura de un marco clase [Fikes et al., 85] que se ejecutan al acceder o almacenar algún valor en una ranura de un marco instanciado de la BC. Así, se puede solicitar la ejecución de los procedimientos para:

a) Leer o almacenar un valor en una ranura [Fikes et al., 85].

a) Anular el valor almacenado en una ranura [Winston, 92].

b) Mantener restricciones. Para Winston [Winston, 92] las restricciones tienen como finalidad asegurar que un cambio en el valor de una ranura se refleja correcta y automáticamente en el valor de otra.

c) Tratar perspectivas y contextos [Winston, 92]. Por ejemplo, la *altura* de un *enano* es *baja* si se considera que un *enano* es una *persona*, pero, el *enano* puede ser *alto* si se le compara con otros *enanos.*

d) Computar dinámicamente valores para obtener información explícita a partir de información implícita almacenada en la BC [Fikes et al., 85].

e) Regular el sistema. Cuando una rutina se ejecuta tiene el control del sistema y esto puede ser utilizado para regularlo [Steinheiser, 90].

f) Los valores activos lanzan la ejecución de un conjunto de reglas cuando un valor concreto se ha almacenado en una ranura [Steinheiser, 90].

g) Manejo de errores [Frost, 86].

Los demonios introducen una gran cantidad de conocimiento procedimental que trabaja sobre el conocimiento declarativo de la representación expresado en la jerarquía de marcos, concretamente, en cada marco clase. La herencia permitirá que, junto con la descripción de las propiedades, estos demonios sean heredados por los marcos de niveles inferiores.

Los procedimientos más utilizados realizan las siguientes funciones:

* Procedimientos que se ejecutan cuando se necesita conocer el valor de una propiedad en un marco instanciado, es decir, cuando la ranura está vacía y se desea conocer el dato. Entonces, el procedimiento rellenará la ranura vacía del marco instaciado con un valor que le haya dado el usuario, o con un valor que obtiene a partir de datos implícitamente almacenados en la BC.

* Procedimientos que se ejecutan cuando se modifica el valor de una propiedad en un marco instanciado. El nuevo dato puede reemplazar valores ya almacenados en la ranura de la instancia cuando se creó la BC o bien introducidos en tiempo de ejecución.

* Procedimientos que se ejecutan cuando se añade un dato en una propiedad de un marco instanciado.

* Procedimientos que se ejecutan cuando se elimina un dato en una propiedad de un marco instanciado.

* Procedimientos que se ejecutan cuando se desea tratar una instancia desde una perspectiva o punto de vista concreto [Winston 92].

* Procedimientos que se ejecutan al razonar sobre un contexto diferente al actual [Winston, 92].

* Procedimientos que comprueban restricciones y determinan si los valores que toma una propiedad son correctos. Para ello se debe comprobar que el número de valores está en el intervalo dado por la cardinalidad mínima y máxima y que el valor está incluido en el rango de valores permitidos. [Fikes et al., 85]

A su vez, los demonios pueden ser de dos tipos [Barr et al., 82]: dirigidos por las metas o por los datos. Estos procedimientos toman el control cuando ciertos sucesos o datos ocurren en la BC.

a) Los *Demonios dirigidos por las Metas* deducen valores de la ranura a partir de valores almacenados en otras, utilizando un complejo, pero eficiente, mecanismo de encadenamiento hacia detrás.

b) *Demonios dirigidos por los Sucesos* que ejecutan las acciones adecuadas cuando el valor de una ranura cambia. Este tipo de demonio se utiliza cuando se razona con encadenamiento hacia adelante.

2.2.3.3.2 Métodos

Los métodos [Fikes et al., 85] son procedimientos asociados a ranuras de marcos clase que se ejecutan como respuesta a un mensaje enviado a dicha ranura. El valor que devuelve al ejecutarse el procedimiento es el valor de la ranura.

Cada ranura que almacena un método tiene un nombre. Los mensajes enviados a los marcos deben especificar la ranura a la que va destinada el mensaje y los argumentos con los que debe ejecutarse el procedimiento o método almacenado en la ranura. Los métodos son heredados, como cualquier otra ranura, por marcos de niveles inferiores.

2.2.3.4 Clasificación de las Inferencias

Las inferencias que se realizan sobre un Sistema Basado en Marcos, Frost [Frost, 86] las clasifica en: de Existencia, de Propiedades Genéricas, de Valores por Omisión, en Reconocimiento de Situaciones Anómalas y por Analogía.

a) De Existencia

Cuando se realiza una equiparación entre una entidad y un marco, el sistema es capaz de inferir la existencia de una entidad del tipo de entidad representado por el marco. La certeza de estas inferencias dependen del VE descrito anteriormente. De igual forma que la Teoría de la Probabilidad se aplica a los Sistemas de Producción, se podría utilizar en los SBM.

b) De Propiedades Genéricas

Cuando una equiparación se realiza entre una entidad E y un marco M, entonces, el sistema puede inferir que E tiene todas las propiedades genéricas asociadas a M. La certeza de estas inferencias dependen del VE entre E y M. Frost [Frost, 86] piensa que los métodos utilizados en Sistemas de Producción para razonar con incertidumbre deberían utilizarse en los marcos.

c) De Propiedades por Omisión

Cuando una equiparación se realiza entre una entidad E y un marco M, y el valor de alguna ranura R en E no es conocida, entonces, si existe algún valor por omisión para R en M, el sistema inferirá que E tiene ese valor en la ranura R. En este caso, la certeza de la inferencia dependerá del VE y de la certeza asociada con los valores por omisión.

d) En Situaciones Anómalas

La ausencia de un valor o la presencia de un valor inapropiado en una ranura de una entidad E, puede significar una situación no habitual.

e) Por Analogía

El *razonamiento por analogía* utiliza sentencias del tipo "X es como Y" para inferir valores de las ranuras de X a través de los valores de las ranuras de Y. Las heurísticas que Frost [Frost, 86], derivadas de las presentadas por Rich [Rich, 83], destaca en este tipo de inferencias son heurísticas para seleccionar valores desde Y y heurísticas para filtrar estos valores.

Para seleccionar valores desde Y:

1. Seleccionar valores que sean relativamente altos o bajos comparados con los valores con los que usualmente se rellena esta ranura.

2. Seleccionar aquellas ranuras que críticamente son importantes en el proceso de equiparación.

3. Seleccionar ranuras que no aparecen en marcos relacionados con Y mediante relaciones similares o fraternales.

4. Utilizar todos los valores de Y.

Para filtrar los valores:

1 Seleccionar ranuras que no están rellenas en X.

2. Seleccionar, entre las ranuras anteriores, aquellas que están normalmente rellenas.

3. Seleccionar el resto de ranuras que no se han seleccionado en el paso 1.

2.2.4 AMPLIACIONES AL CONCEPTO DE MARCOS

2.2.4.1. MetaClases

El enfoque tradicional de marcos considera a los marcos clase como clases de individuos y a los marcos instanciados como elementos individuales de las clases. En este nuevo enfoque, si un marco instanciado representa un conjunto de elementos, los elementos del conjunto son los valores que toma una propiedad que referencia al conjunto de individuos que forman la instancia. Por ejemplo, el *Real Madrid* es una instancia de la clase *Equipos de Fútbol de Primera División* y va a heredar todas las propiedades definidas en esta clase y en todas las clases con las que esta clase esté unida. A su vez, el *Real Madrid* representa un conjunto de personas que son sus jugadores. Para representar a los jugadores de este equipo se utilizaría una ranura llamada *Jugadores* que se rellena con el conjunto de jugadores que juegan en el *Real Madrid*, tal y como se muestra en la figura 2.12.

El problema surge al representar el *Real Madrid* como la clase que representa al conjunto de personas que juegan en el Real Madrid en vez de representarlo como una instancia de la clase *Equipos de Fútbol de Primera División*, siendo sus instancias los jugadores que juegan en el *Real Madrid*. En este caso, el *Real Madrid* clasificaría a los *Jugadores del Real Madrid* y heredaría las propiedades definidas en el marco *Jugadores de Fútbol de Primera División* y en los marcos con los que este marco estuviera conectado. Sin embargo, esta representación impide que el marco *Jugadores del Real Madrid* herede las propiedades definidas en el marco *Equipos de Futbol de Primera División*. Por tanto, el problema que presenta esta representación es que se pueden heredar atributos sobre los elementos de la clase pero no sobre la clase en sí, tal y como se muestra en la figura 2.13.

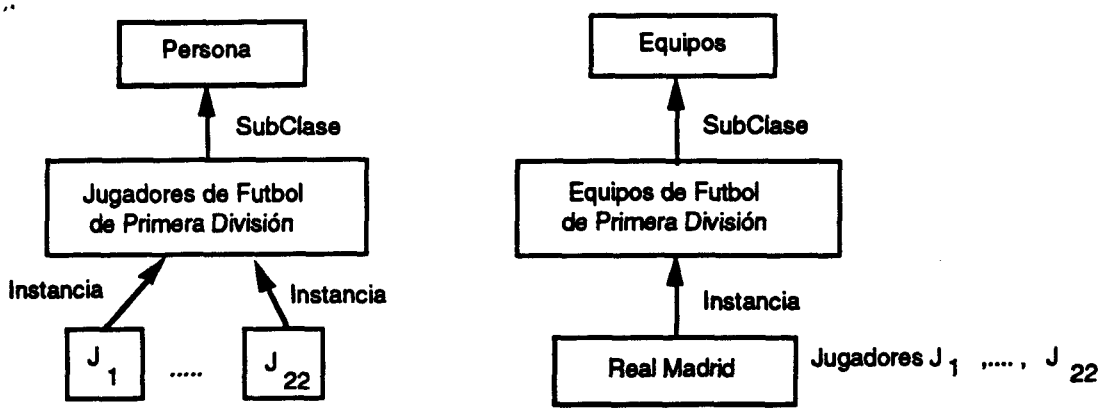


Figura 2.12. Representación tradicional de un conjunto de individuos

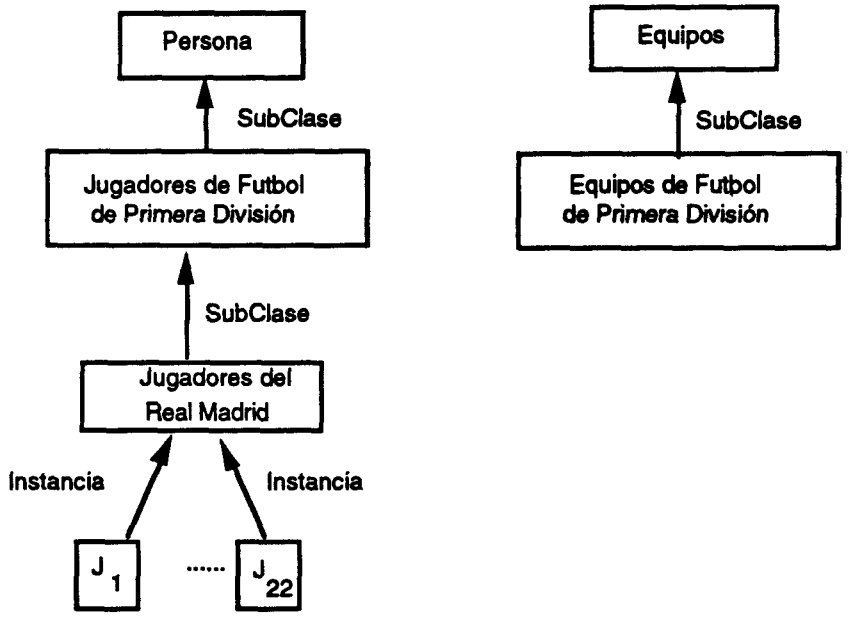


Figura 2.13. Representación del *Real Madrid* como un conjunto de jugadores.

El problema consiste en que la instancia *Real Madrid* representa un conjunto de individuos y, por tanto, los individuos, es decir, sus *jugadores*, podrán heredar propiedades comunes al conjunto de individuos, representados por la instancia *Real Madrid*, convirtiéndose la instancia *Real Madrid* en una clase instanciada cuyas propiedades pueden, o no, pertenecer a las instancias individuales. Rich y Knight [Rich et al., 91] resuelven este problema utilizando las Metaclases.

Stefik y Bobrow [Stefik et al., 86] y Rich y Knight [Rich et al., 91] definen las **Metaclases** como clases cuyos elementos (instancias) son a su vez clases. Una clase podrá estar entonces relacionada con otra clase mediante una relación de

especialización y con una metaclase mediante una relación de instanciación. La clase, por ser un conjunto, puede almacenar propiedades que sus elementos poseen, utilizándose la herencia de propiedades para inferir propiedades sobre el conjunto. Pero, al ser la clase un elemento, posee un conjunto de propiedades que no pertenecen a las instancias individuales, sino a la clase como tal.

La resolución del problema, anteriormente mencionado, es la que se muestra en la figura 2.14, donde *Equipos de Futbol de Primera División* es una metaclase.

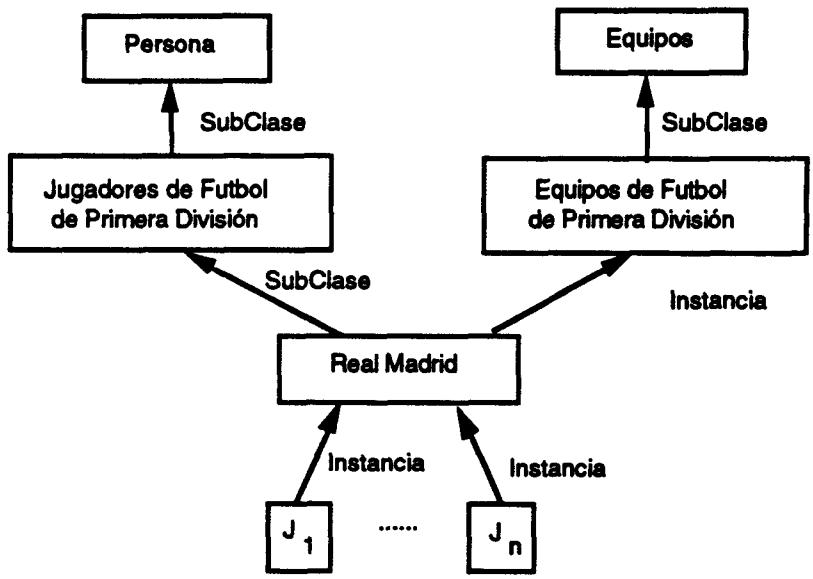


Figura 2.14. Representación del *Real Madrid* y sus jugadores utilizando MetaClases

2.2.4.2 Prototipos

El término **Prototipo** se ha utilizado en IA y en Psicología Cognoscitiva para expresar individuos estereotipados o individuos de una clase que se utilizan como referencia para describir otros individuos por comparación [Bobrow et al.,77].

El uso explícito de marcos prototipos en la construcción de Sistemas Basados en Marcos ha sido una extensión al formalismo de marcos realizada por Nado y Fikes [Nado et al., 92]. Hasta entonces, se han descrito prototipos utilizando un conjunto de ranuras de instancia definidas dentro de los marcos clase. Para Nado y Fikes, un prototipo es una parte de un marco clase que describe arbitrariamente miembros de la clase y tiene asociado un conjunto de ranuras llamadas ranuras prototipo. Cada prototipo tiene un nombre que se puede utilizar en la descripción del prototipo como

una variable cuantificada universalmente para referirse a un miembro cualquiera de la clase.

Cuando un valor de una ranura prototipo, en el marco prototipo asociado a un marco clase, se hereda por la ranura correspondiente en un marco instanciado o en un prototipo de una subclase, cualquier referencia en el valor del prototipo se sustituye por referencias al marco instanciado o al nuevo prototipo.

2.2.4.3. Ranuras como Marcos

Dado que un marco se define mediante un conjunto de atributos o ranuras y dado que cada ranura se define utilizando a su vez un conjunto de facetas o atributos, es posible definir, a su vez, cada una de las ranuras que forman parte de un marco como marcos. Dos alternativas, la proporcionada por Rich y Knight y la de Nado y Fikes, permiten utilizar este nuevo concepto.

Para Rich y Knight [Rich et al., 91] una ranura es una relación entre un dominio y un rango. El dominio referencia el marco clase en el que se ha definido la propiedad y el rango se puede dividir en dos apartados: el rango y la restricción del rango. El rango proporciona la clase en la cual los componentes del rango son instancias o elementos y la restricción del rango contiene una expresión lógica que restringe el rango de los elementos que componen el rango. Los marcos que representan ranuras se agrupan en una jerarquía de marcos que tiene, como marco raíz de la jerarquía, a la metaclase "Ranura". En esta metaclase se definen como propiedades de instancia, además de las ranuras dominio, rango y restricción del rango, todas las facetas predefinidas y aquellas definidas "ad hoc" en cada uno de los marcos que componen el Sistema Basado en Marcos. Además, se definirán las ranuras subclase e instancia, ranuras necesarias para construir la jerarquía de marcos asociada a las ranuras del Sistema Basado en Marcos. A su vez, entre marcos ranuras, pueden existir, por ejemplo, relaciones de especialización y por tanto se puede aplicar la herencia de propiedades para realizar inferencias. Junto con la herencia de propiedades, la comprobación y el mantenimiento de consistencias, la herencia de valores por defecto y la computación de procedimientos son técnicas que se deberían utilizar para realizar inferencias. El problema que plantea este modelo es que, aunque el modelo es sencillo y consistente, no es fácil de utilizar. Para simplificar el modelo, sólo se utiliza una única jerarquía de marcos y se introduce en la definición de cada ranura las facetas dominio, rango y restricción del rango.

Para Nado y Kikes [Nado et al., 92] una ranura es un marco clase que aparece como componente de otro marco clase. Una ranura es entonces una entidad relacionada por una relación binaria con la entidad representada por el marco del cual la ranura es un componente. Dado que una ranura es ahora un marco clase, la ranura puede tener asociada prototipos que describen el valor o conjunto de posibles valores que la ranura puede tener. Los miembros de la clase que la ranura representa son los valores de la ranura.

2.2.4.4 Referencias Autocontenidas

Las referencias autocontenidas [Nado et al., 92] permiten que una parte de un marco clase que describe los miembros de dicha clase, contenga referencias a ella misma. Estas referencias, al ser heredadas por los miembros de la clase, se sustituyen por referencias de la clase miembro. La capacidad de introducir referencias autocontenidas, no es algo novedoso introducido por Nado y Fikes en el concepto de marco, pues ya existían implementaciones de marcos que trabajaban con este concepto. Sin embargo, la novedad que los autores proponen es introducir las referencias en los prototipos, teniendo así referencias autocontenidas de prototipos y aumentando así la expresividad del formalismo.

2.2.4.5 Vectores de Ranuras

Block y Chan [Block et al., 89] amplian la semántica de los Marcos al permitir expresar estructuras de datos complejas, concretamente vectores, como ranuras. Por ejemplo, supóngase que se tiene un marco que representa el conjunto de personas que hay en un aula y que la ranura *Estudiantes* contiene el conjunto de estudiantes de dicha aula almacenados en una lista. Añadir o borrar un estudiante implica manipular la lista. La solución que Block y Chan proponen es definir un nuevo tipo de ranura en la que el nombre de la ranura lleva asociado una variable que representa la dimensión del vector. Una instancia tendrá tantas ranuras de este tipo como elementos se hayan definido en el vector. En este enfoque, para manipular la lista, solamente hay que operar a nivel de ranura.

2.2.5 RESUMEN DE LA TERMINOLOGIA EMPLEADA

En la tabla 2.1 se presenta el resumen de la terminología utilizada entorno al concepto de Marco. La primera entrada de la tabla se corresponde con la palabra española adoptada en este trabajo para designar los conceptos que aparecen en inglés. La segunda entrada de la tabla es la definición del concepto.

TERMINO	DEFINICION
Marco Frame *(Minsky)*	Estructura de datos que representa situaciones estereotipadas.
Marco Clase Class Frame	Conjunto de entidades individuales que comparten las mismas propiedades funcionales o estructurales.
Marco Instancia Instance Frame	Entidad concreta de un tipo representado por un marco clase.
Marcos Alternativos Alternative View Frames	Marcos que representan conceptos, clases, o entidades desde varios puntos de vista.
Sistema de Marcos Frame System	Conjunto de Marcos conectados entre sí.
Relación SubClase ISA Relation *(Frost)* Subset Relation *(Rich, Knight)* SuperClass Link *(Reichgelt)* A Kind of Link *(Winston)*	Existe una Relación SubClase entre A y B si A representa un conjunto de entidades que es un subconjunto del conjunto de entidades representadas por B.
Relación SuperClase SubClass Relation *(Frost, Rich)* Sub Link *(Reichgelt)* Class Link *(Winston)*	Relación inversa de la Relación SubClase.
Relación Instancia Instance Relation Element of Relation *(Rich)* Set Membership Relationship *(Frost)* Instance Link *(Reichgelt)*	La Relación Instancia entre dos marcos A y B se utiliza para expresar que el marco A es una instancia del marco clase B.
Relación Elementos_Clase All-Instance Relation *(Rich)* Member of Link *(Reichgelt)*	Relación inversa de la Relación Instancia.
Relación Fraternal Sibling Relationship *(Frost)*	A y B pueden están relacionados mediante una Relación Fraternal si tanto A como B tienen como padre el mismo marco C.

Relación Disjunta Disjoint Relationship *(Frost)* Mutually disjoint with Relation *(Rich, Knight)*	Entre dos marcos A y B existe una Relación Disjunta, si tanto A como B representan conjuntos disjuntos de entidades.
Relación No Disjunta Non Disjoint Relationship *(Frost)*	Dos marcos pueden relacionarse mediante la Relación No Disjunta, si representan conjuntos no disjuntos de entidades.
Relación Similar Similar Relationship *(Frost)*	Dos marcos se pueden relacionar mediante la Relación Similar, si representan conjuntos de entidades cuyos miembros tienen propiedades en común suficientes para ser considerados como similares.
Relación Recubrimiento Is Coverd by Relation *(Rich, Knight)*	La Relación Recubrimiento relaciona una clase con el conjunto de sus subclases, de tal forma que la unión de dichas subclases es la clase inicial.
Ranura Slot	Propiedad del marco.
Propiedad de Clase Own Slot *(Fikes, Kehler)*	Propiedades genéricas de un concepto o clase. Estas propiedades toman siempre el mismo valor en todos los elementos del conjunto
Propiedad de Instancia Member Slot *(Fikes, Kehler)* Prototype Slot *(Nado, Fikes)*	Propiedades comunes a todos los elementos del concepto o clase que el marco representa y que se rellenan en las instancias.
Faceta Facet *(Fikes, Kehler)*	Atributo de la ranura.
Equiparación Matching	Determina el marco clase que mejor describe la situación actual.
Valor de Equiparación Mach Value *(Frost)*	Grado de idoneidad de la equiparación.
Herencia Inheritance *(Touretzky)*	Aplicar información general a la situación actual concreta. Transferencia de propiedades entre marcos utilizando únicamente las relaciones SubClase e Instancia.
Herencia Simple Simple Inheritance *(Touretzky)*	Existe un único camino en el que se aplica la herencia que une la instancia con el marco raíz de la jerarquía.
Herencia Multiple Multiple Inheritance *(Touretzky)*	Existen varios caminos en los que se aplica la herencia que une la instancia con el marco raíz de la jerarquía.
Herencia Polar Unipolar Inheritance *(Touretzky)*	Herencia que permite la transferencia de conocimiento de sentencias no negativas, es decir, positivas.
Herencia Bipolar Bipolar Inheritance *(Touretzky)*	Herencia que permite la transferencia de conocimiento de sentencias positivas y negativas.
Herencia Monótona Monotonic Inheritance *(Touretzky)*	Herencia que no permite trabajar con excepciones.

Herencia No Monótona Nonmonotonic Inheritance *(Touretzky)*	Herencia que permite excepciones.
Herencia Homogénea Homogeneous Inheritance *(Touretzky)*	Herencia permanentemente monótona o permanentemente no monótona.
Herencia Heterogénea Heterogeneous Inheritance *(Touretzky)*	Herencia que en ocasiones es monótona y en otras es no monótona.
Herencia No Multivaluada Single-Valued Inheritance *(Touretzky)*	Herencia que permite que una propiedad se rellene con un único valor.
Herencia Multivaluada Multivalued Inheritance *(Touretzky)*	Herencia que permite que una propiedad se rellene con más de un valor.
Demonios, Valores Activos, Disparadores Demons, Active Value, Triggers *(Barr)*	Procedimientos que se ejecutan al acceder, almacenar o borrar un dato en una propiedad de un marco instanciado.
Métodos Methods *(Fikes)*	Procedimientos asociados a ranuras que se ejecutan cuando la ranura recibe un mensaje de activacción.
MetaClase MetaClass*(Rich, Knight)*	Clase cuyas Instancias son conjuntos de elementos.
Prototipo Prototype *(Nado, Fikes)*	Parte de un marco Clase que describe arbitrariamente miembros de la clase pero no la clase en sí.

Tabla 2.1. Resumen de la terminología de Marcos

2.3. OBJETOS

2.3.1 LA ORIENTACION A OBJETOS

Mientras que Minsky [Minsky, 75], Schank [Schank, 73], Bobrow [Bobrow et al., 77] y otros utilizaban el Paradigma de los Marcos para representar conocimiento en los años setenta, otro grupo de investigadores tomaron como alternativa el Paradigma de los Objetos. Alonso [Alonso, 93] define el Paradigma de los Objetos como:

> *Una filosofía de desarrollo y empaquetamiento de software que permite crear unidades funcionales extensibles y genéricas, de*

forma que el usuario pueda aplicarlas según sus necesidades y conforme con las especificaciones del sistema a desarrollar.

Koshafian y Abnous [Koshafian et al., 90] definen la Orientación a Objetos, en adelante OO, como:

Una disciplina de modelización de software que permite construir fácilmente sistemas complejos a partir de componentes elementales.

Snyder [Snyder, 93] define los Objetos como:

Una entidad que proporciona un servicio a sus clientes. El número de clientes depende del sistema, pero, en general, un cliente puede ser una persona o un programa y, un servicio es una actividad que se realiza como respuesta a la petición de un cliente.

La modelización y representación intuitiva del mundo real ha hecho que esta metodología de diseño y desarrollo sea la que más auge está teniendo en la construcción del software. En gran medida, este auge se debe a que los Objetos captan de manera natural la realidad y la orientación a objetos crea modelos muy cercanos a ella [Khoshafian et al., 90]. En palabras de Ledbetter y Cox [Ledbetter et al., 85]:

La programación orientada a objetos permite una representación más directa del modelo del mundo en el código. El resultado es que la transformación de los requisitos del sistema definidos por el usuario, en una especificación del sistema definido en términos de la máquina, se reducen considerablemente.

o, en palabras de Coad y Yourdon [Coad et al., 91]:

El Diseño Orientado a Objetos tiene, como una de sus principales motivaciones, identificar las clases y los objetos que acerquen la representación del sistema a la vista conceptual del problema y a su posterior implementación.

Actualmente, la tecnología Orientada a Objetos se ha incorporado como idea básica en muchas áreas de investigación en la Ciencia de la Computación. Diferentes tecnologías estan convergiendo tomando como eje la tecnología de objetos. Algunas de estas áreas son: Arquitectura de Computadores, Sistemas Operativos, Ingeniería del

Conocimiento, Sistemas de Bases de Datos, Interfaces de Usuarios, etc. [Jacobson, 93]. El uso de esta tecnología por otras áreas diferentes a la Ingeniería del Software se debe a la calidad del software con ella construido, calidad que se expresa en función de factores externos e internos [Meyer, 88]. Como factores externos, perceptibles por un usuario o cliente, se tiene un software: correcto, robusto, ampliable, reutilizable y compatible. Como factores internos, destacan los diseños descentralizados y flexibles, con módulos coherentes e interfaces bien definidas, que aseguran la reutilización, la ampliación y la compatibilidad del software desarrollado.

El Diseño Orientado a Objeto (DOO) basa la estructura del sistema en las clases de objetos que manipula. Así, en un diseño orientado a objetos, inicialmente no interesa conocer cómo el sistema lo hace, sino con qué objetos lo hace, dejando para los últimos pasos la decisión de determinar las funciones, en caso de haberlas. Para satisfacer los requisitos de reutilización y para hacer a los programas complejos más manejables, un principio básico del DOO es ocultar información, ocultación que se consigue utilizando las siguientes técnicas [Snyder, 89]:

Abstracción: Consiste en suprimir detalles.

Modularidad: Consiste en descomponer un programa en subprogramas, con interacciones limitadas y bien definidas.

Encapsulación: Consiste en prohibir accesos a la información.

2.3.2 LENGUAJES ORIENTADOS A OBJETOS

Khoshafian y Abnous [Khoshafian et al., 90] clasifican los lenguajes de programación orientados a objetos en:

* Ampliaciones, dialectos y versiones de *Smalltalk*
* Lenguajes convencionales con ampliaciones orientadas a objetos
* Lenguajes orientados a objetos fuertemente "tipados"
* Ampliaciones de LISP orientadas a objetos

Ampliaciones, dialectos y versiones de *Smalltalk*

El primer lenguaje de programación orientado a objetos, *Simula* [Dahl et al., 66], se construyó a finales de los años sesenta y principios de los setenta. Desde entonces, se han construido múltiples lenguajes orientados a objetos, entre los que caben destacar *Smalltalk*. *Smalltalk*, durante los años setenta fue un proyecto de investigación de IA en Xerox Palo Alto Research Park. *Smalltalk-72, -74, -76, -78, 80* y, más recientemente, *Smalltalk/V* [Goldberg et al., 83], son versiones de este lenguaje.

Lenguajes convencionales con ampliaciones orientadas a objetos

Hoy en día, la mayoría de los lenguajes de programación convencionales incluyen ampliaciones orientadas a objetos. Prueba de ello, son los dialectos orientados a objetos de C: *C++* [Stroustrup, 86] y *Objective-C* [Cox, 87]. Para el lenguaje Pascal, en entorno Macintosh, se tiene la extensión *Object Pascal* [Schmucker, 86] y, *Turbo Pascal*, para computadoras personales.

Lenguajes orientados a objetos fuertemente "tipados"

Lenguajes menos convencionales, fuertemente "tipados" y orientados a objetos, son *Eiffel* [Meyer, 88], lenguaje que introdujo en la Orientación a Objetos los conceptos de precondiciones y postcondiciones y *Trellis/Owl* [Schaffert et al., 86].

Ampliaciones de LISP orientadas a objetos

LISP también incluye ampliaciones orientadas a objetos. Ejemplos de estas extensiones son: *SCOOPS* desarrollado por Texas Instrument y el MIT, *Symbolics Flavors* evolución del lenguaje de Flavors [Moon, 86], *Common LOOPS* desarrollado en Xerox Parc y, por último, *CLOS* [Keene, 88], abreviatura de Common LISP Object System, que es el lenguaje base para crear el estándar ANSI de Common LISP.

Para Snyder [Snyder, 93], una consecuencia inmediata, fruto del gran número de lenguajes, es la ausencia de un vocabulario común a todos ellos y la presencia de una gran variedad de términos y definiciones orientados a objetos. Para Jacobson [Jacobson, 93] esta falta de uniformidad ocurre no sólo en los lenguajes de programación, sino también en las técnicas y métodos orientados a objetos. Así, múltiples términos se utilizan para describir el mismo concepto y el mismo término puede utilizarse con diferentes significados. Snyder [Snyder, 93] agrupa las

definiciones en tres secciones, y para cada término de cada sección muestra cómo diferentes lenguajes orientados a objetos llaman a cada concepto. Un glosario de los terminos más utilizados fue también realizado por Stefik y Bobrow [Stefik et al., 86]. Se pasan a analizar los conceptos que aparencen en la OO.

2.3.3 CONCEPTOS DE ORIENTACION A OBJETOS

Al analizar los elementos fundamentales básicos que configuran el Paradigma de Objetos, se observa que cada autor utiliza diferentes enfoques para exponer el paradigma [Alonso, 93].

Así, Bertrand Meyer [Meyer, 88] distingue los siguientes puntos: Estructura modular basada en objetos, Abstracciones de datos, Gestión automática de memoria, Tipos, Herencia, Polimorfismo y Enlace dinámico y, por último, Herencia múltiple y repetida. Para Khoshafian [Khoshafian et al., 90], los tres aspectos más importantes del PaOO son: los Tipos abstractos de datos, la Herencia y la Identidad de los objetos. Y, para Winblad, Edwards y King [Winblad et al., 90] los conceptos claves son: Encapsulación, Abstracción, Polimorfismo y Persistencia.

2.3.3.1 Tipos Abstractos de Datos

Los tipos abstractos de datos, en adelante TAD, son una ampliación de los tipos de datos que aparecen en los lenguajes de programación convencionales. Khoshafian [Khoshafian et al., 90] define los TAD como una estructura de datos y sus operaciones asociadas.

Para Meyer [Meyer, 88] la especificación de un TAD describe una clase de estructuras de datos no por su implementación, sino por una lista de servicios disponibles en la estructura de datos y por las propiedades formales del servicio. Para Meyer, la especificación de un TAD consta de los siguientes elementos:

Tipo Abstracto de Datos = Tipos + Funciones + Precondiciones + Axiomas

donde:

Tipos: Conjunto de tipos utilizados en la especificación del TAD.

Funciones: Conjunto de servicios disponibles en las instancias del TAD.

Precondiciones: Conjunto de requisitos que deben cumplirse para aplicar una Función.

Axiomas: Conjunto de restricciones al TAD.

Mientras que las dos primeras se utilizan para representar sintácticamente el TAD, las Precondiciones y los Axiomas representan su semántica.

Entre las ventajas que los TAD proporcionan, Khoshafian y Abnous [Khoshafian et al., 90] destacan las siguientes:

* Mejora en la conceptualización y modelización del mundo real.
* El software es más robusto.
* Mejora del rendimiento.
* Captura mejor la semántica del tipo.
* Separa la implementación de la especificación.
* Hace a los sistemas ampliables.

En la mayoría de los lenguajes de Programación Orientada a Objetos (POO) [Khoshafian et al., 90], el término que se utiliza para nombrar la implementación del TAD es el de **clase**. Las clases [Meyer, 88] se utilizan para describir un conjunto de estructuras de datos caracterizadas por tener las mismas propiedades. Los elementos pertenecientes a la clase reciben el nombre de **instancias de la clase u objetos**. La diferencia entre una clase y un objeto es la misma que la que hay entre un conjunto y un elemento perteneciente al conjunto.

Las clases incorporan [Khoshafian et al., 90], además del nombre de la clase y de las operaciones definidas en el interfaz, la representación interna de la estructura de datos y la implementación de las operaciones definidas en el TAD.

a) La estructura de datos se expresa utilizando variables que representan atributos. Las variables son de dos tipos: Variables de Clase y Variables de Instancia. Las **Variables de Clase** [Stefik et al., 86], en inglés *Class Variables*, son variables que están almacenadas en la clase y cuyo valor se comparte por todas las instancias de la clase. Las **Variables de**

Instancia [Stefik et al., 86], en inglés *Instance Variables*, son aquellas que toman valores diferentes en cada instancia u objeto. Para Snyder [Snyder, 89], el conjunto de variables de instancia representan el estado interno del objeto y muestran cómo va variando con el tiempo la conducta del sistema, no pudiéndose acceder desde fuera del objeto al estado interno del objeto pues éste está encapsulado. Recientemente, el mismo autor [Snyder, 93] ha propuesto utilizar el término **Variable de Estado**, en inglés *State Variable*, porque pueden existir objetos que no sean instancias de ninguna clase.

b) La definición de la Clase especifica también los **Métodos**. Un método [Snyder, 93] es un procedimiento que realiza un servicio. El servicio abarca tanto a la lectura o escritura de variables de estado como a la realización de operaciones más complejas que impliquen a otros objetos y métodos. En la OO se accede a una *Estructura de Datos* de un TAD utilizando los procedimientos o *métodos* que en el TAD se han definido. A este concepto se le llama **Encapsulación**, en inglés *Encapsulation*. En cualquier caso, siempre existe un método por cada operación que la clase soporta.

Un mismo método puede ser solicitado por objetos de distinto tipo. En este caso, dicho método tiene asociadas diferentes semánticas e implementaciones, se dice entonces que se tiene un método "sobrecargado", en inglés **Overleading Method**. Si la elección de la implementación a ejecutar se selecciona en tiempo de ejecución, se tiene un **Enlace Dinámico**, en inglés *Dynamic Binding* o *Late Binding*, si es en tiempo de compilación, se tiene un **Enlace Estático**, en inglés *Static Binding* o *Early Binding*. Independientemente de cómo se ejecute el procedimiento, el método comienza a ejecutarse cuando un objeto recibe un **Mensaje** de un cliente que solicita un servicio.

El mismo mensaje puede realizar diferentes acciones dependiendo del objeto que reciba el mensaje. A este fenómeno se le llama **Polimorfismo** [Winblad et al., 90].

Cada objeto tiene un **Protocolo** [Khoshafian et al., 90] o conjunto de mensajes a los cuales el objeto puede responder. El protocolo de un

objeto define su **Interfaz** [Khoshafian et al., 90]. El interfaz es el conjunto de métodos definidos para las instancias de la clase.

Las clases se conectan con otras clases mediante las relaciones *Cliente* y *Descendiente* [Meyer, 88]. Una Clase A es *Cliente* de B cuando A utiliza los procedimientos definidos en B. Sin embargo, una clase A es *Descendiente* de B cuando A es una especialización de B. La relación de descendencia origina la herencia, es decir, el mecanismo que permite a objetos y clases compartir estructuras de datos y procedimientos definidos en clases de niveles superiores de la jerarquía.

Aunque las clases, los métodos y las variables son los conceptos principales que aparecen en la POO, no hay que olvidar otros términos como: MetaClase y Perspectivas.

MetaClase *(MetaClass)*

Se refiere a una clase cuyas instancias son clases [Stefik et al., 86].

Perspectivas *(Perspectives)*

Es una forma de componer objetos, interpretando la composición como diferentes vistas de la misma entidad conceptual. Cada perspectiva ofrece diferentes vistas del objeto, con sus variables y métodos.

2.3.3.2 Herencia

La idea de herencia apareció en los primeros lenguajes orientados a objetos como *Simula* [Dahl et al., 66] y *Smalltalk-80* [Goldberg et al., 83]. En estos sistemas, se creaban nuevas clases utilizando las definiciones de clases ya existentes. La nueva clase heredaba las variables y métodos de las viejas clases y añadía sus variables y métodos propios. Esta relación de herencia entre clases pronto dió lugar a otro tipo de relación [America, 89], relación de especialización que se define como: *Si la clase B hereda de la clase A, cada instancia de la clase B tendrá al menos las variables y métodos que las instacias de la clase A tienen.* Este enfoque considera a las instancias de la clase B como especializaciones de las instancias de la clase A y convierten a B en una *subclase* de A. Para America [America, 89] la primera jerarquía descrita se basa en la compartición del código, o descripción de la estructura interna de los objetos, que coincide plenamente con otra jerarquía, la que lleva asociado el uso de los objetos y, por consiguiente, su conducta observable desde el exterior.

Dado que la compartición de código no siempre lleva a una especialización en la conducta, Snyder [Snyder, 86] y America [America, 87] propusieron construir dos jerarquías: una para el código y otra para la conducta. Por ejemplo, al añadir un nuevo método a un conjunto de variables y métodos, puede ocasionar variaciones en la conducta del método viejo, pues puede haber compartición de código pero no especialización en la conducta. Por otra parte, también es posible obtener la misma funcionalidad con diferentes representaciones. Por ejemplo, un número complejo se puede representar en coordenadas polares o en coordenadas cartesianas, y una pila se puede representar como una lista o como un vector. Suponiendo que ambas representaciones del número complejo y de la pila ofrecen los mismos servicios, la funcionalidad o conducta alcanzada es la misma, aunque se utilicen diferentes implementaciones.

Por tanto, el término herencia en POO es un término ambigüo [Snyder, 89], que se utiliza tanto para expresar la estructura interna del objeto centrada en la compartición del código de las variables y de los métodos, como para expresar la conducta del objeto visible desde el exterior. Al tipo de herencia que se centra en la compartición de código Snyder [Snyder, 89] y America [America, 89] la llaman respectivamente *Herencia de Implementación* o *Herencia*; y a la herencia centrada en la conducta o funcionalidad presentada por una clase al recibir un mensaje, los mismos autores la llaman *Herencia de Especificación* o *Subtipado*. La herencia de especificación se relaciona con la herencia de implementación [Snyder, 89] en que la herencia de implementación puede utilizarse para definir una nueva clase que es un subtipo de la clase original. Mientras que Snyder se centra en la herencia de implementación, America describe una definición de tipos y subtipos que son independientes tanto de la representación interna de los objetos como de la herencia de implementación.

Para America [America, 89], un **tipo** es un conjunto de objetos que tienen algunas propiedades intrínsecas (la propiedad no cambia durante la vida del objeto) en común con una conducta externamente observable. El tipo, es la *especificación* de la conducta de sus elementos u objetos, especificación que comprende el nombre de los métodos que los objetos deberían tener, los tipos de parámetros y resultados de dichos métodos, y el estado bajo el cual un mensaje debe ser enviado a un objeto, así como los posibles valores que puede dar como resultado.

Por otro lado, la herencia de implementación se utiliza [Snyder, 89] para definir una nueva clase, llamada clase hijo o subclase, como una modificación gradual de una clase ya existente llamada clase, superclase o padre. La definición del hijo puede:

* Aumentar la representación del padre, añadiendo más declaraciones de variables de instancia.

* Añadir nuevas operaciones.

* Sustituir operaciones, proporcionando nuevas definiciones.

* Ocultar operaciones definidas en el padre que no aparecen en el interfaz del hijo.

* Refinar o borrar declaraciones de variables de instancia.

2.3.3.3 Características de la Orientación a Objetos

Snyder [Snyder, 93], tomando como punto de partida diferentes sistemas orientados a objetos, extrae las características comunes a todos ellos y así propone el nucleo de conceptos básicos que deben estar presentes en cualquier lenguaje orientado a objetos, conceptos que se materializan en los siguientes once puntos:

1. *Todos los objetos incorporan explícitamente una abstracción* y, por consiguiente, tienen un significado.

2. *Los objetos proporcionan servicios.* La abstracción asociada a un objeto se materializa en un conjunto de servicios que los clientes solicitan.

3. *Los clientes solicitan peticiones.* Los clientes plantean peticiones de servicios a los objetos. Al cliente, solamente le interesa que la petición solicitada se realice. Al aislar al cliente de los detalles de la implementación, se puede modificar la implementación del objeto sin modificar al cliente.

4. *Los objetos están encapsulados.* La única forma que tienen los clientes de acceder a los objetos es mediante peticiones de servicios.

5. *Las peticiones indentifican operaciones.* Una petición lleva indicado un servicio y, por consiguiente, la operación que se va a realizar.

6. *Las peticiones pueden identificar objetos.* Una petición puede llevar asociada unos parámetros y el servicio puede devolver uno o más resultados. Un parámetro o un resultado puede identificar un objeto. En

la mayoría de los sistemas, cada petición tiene asociados unos parámetros que identifican el objeto que realiza la petición.

7. *Se pueden crear nuevos objetos.* Un cliente puede solicitar la creación de nuevos objetos diferentes a los ya existentes.

8. *Las operaciones pueden ser genéricas.* Un servicio puede tener diferentes implementaciones (diferente código) para diferentes objetos, los cuales, pueden tener o no conductas observables diferentes. Si la implementación a ejecutar se selecciona en tiempo de ejecución, se tiene un *enlace dinámico*, en caso contrario, se tiene un *enlace estático.*

9. *Los objetos se pueden clasificar en función de sus servicios.*

10. Los *objetos pueden tener implementaciones comunes*, tanto en el formato de los datos, como en el código que realiza los servicios.

11. *Los objetos pueden compartir implementaciones parciales.* Los objetos con conductas similares comparten parte de sus implementaciones y parte no.

Se pueden establecer algunas restricciones [Khoshafian et al., 90] sobre el TAD con el fin de preservar su semántica y así construir sistemas completos y correctos, bien estableciendo restricciones en los objetos y en las variables instanciadas o asociando precondiciones y postcondiciones en los métodos. Ejemplo de las primeras son rutinas con las que exclusivamente se pueden realizar los accesos y actualizaciones a las variables instancias del objeto y, rutinas que se disparan bien al acceder, o al actualizar las variables instancias.

Las precondiciones y las postcondiciones asociadas a los métodos se refieren a las precondiciones y postcondiciones asociadas a las operaciones del TAD. Eiffel [Meyer, 88] permite el uso de precondiciones y postcondiciones.

2.3.4 PRECONDICIONES Y POSTCONDICIONES

Una de las primeras referencias a la idea de probar programas fue presentada por J. McCarthy en 1961 [McCarthy, 61] y, posteriormente, en el congreso del IFIP en 1962. McCarthy señalaba que, en vez de plantear casos de pruebas a los programas

hasta que son depurados, se debería probar que estos programas cumplen unas propiedades deseadas. Varios años más tarde, en 1966, Naur [Naur, 66] proporcionaba una técnica informal para realizar dicha prueba. Un año más tarde, en 1967, en una reunión de la American Mathematical Society, Floyd [Floyd, 67] introduce los conceptos de: *Aserción, Puntos de Corte e Invariante de Bucle*, en organigramas y sugiere, que la especificación de las técnicas de prueba, podrán proporcionar una definición adecuada de un lenguaje de programación. Hoare [Hoare, 69] recogió la sugerencia de Floyd y, en 1969, definió un pequeño lenguaje de programación en términos de un sistema lógico de axiomas y de reglas de inferencia con el fin de probar la corrección de programas como una extensión al cálculo de predicados. Con el artículo de Hoare nació una escuela centrada en la definición axiomática de lenguajes de programación durante la década de los 70. En 1976, Dijkstra [Dijkstra, 76] escribió un libro en el que introducía el concepto de *Precondición más débil* para un pequeño lenguaje de programación y mostró, a través de unos ejemplos, cómo se podría utilizar este concepto en la prueba de programas. Fué entonces cuando se aclaró cómo realizar los *Invariantes de Bucle*. Durante la década de los 70, múltiples grupos criticaron y apoyaron los inconvenientes y las ventajas de encontrar un invariante para cada bucle. La técnica de prueba de programas se distorsionó por las distintas ideas de la palabra "prueba". De Millo [De Millo et al., 79] escribió un artículo en el que resumía los principales argumentos contra la prueba de programas.

En 1981, Gries [Gries, 81] utiliza las precondiciones y postcondiciones para especificar formalmente programas. La notación {Q} S {R}, donde Q y R son fórmulas lógicas y S un programa, tiene el siguiente significado:

> *Si la ejecución de S comienza en un estado que satisface Q, entonces, se garantiza que S termina siempre en un tiempo finito en un estado que satisface R.*

A Q se le conoce como **Precondición** de S, a R como **Postcondición** de S, S es un programa, y {Q} S {R} es un predicado que puede ser cierto o falso. Obsérvese que no se dice nada sobre la ejecución que comienza en un estado que no satisface Q.

Los conceptos de precondición y postcondición han sido utilizados por la Ingeniería del Software a nivel de diseño y de implementación. Meyer [Meyer, 88] fue el primero en utilizar estos conceptos en el Paradigma de la Orientación a Objetos al introducir estos conceptos en el lenguaje Eiffel de Programación Orientado a Objetos.

2.3.5 RESUMEN DE LA TERMINOLOGIA EMPLEADA

En esta sección se presenta la tabla 2.2 que resume la terminología utilizada alrededor del concepto de objeto. La primera entrada de la tabla se corresponde con la palabra española utilizada para designar los conceptos que aparecen en inglés. la segunda entrada de la tabla es la definición del concepto.

TERMINO	DEFINICION
Objeto Object	Entidad que proporciona un servicio a sus clientes. El número de Clientes depende del sistema, pero, en general, un cliente puede ser una persona o un programa y, un servicio es una actividad que se realiza como respuesta a la petición de un cliente
Tipo Abstracto de Datos Abstract Data Type	Estructura de datos y operaciones asociadas
Clases Class	Describen un conjunto de estructuras de datos caracterizadas por tener las mismas propiedades.
Instancia Instance, Object	Elemento perteneciente a una clase.
Variable de Clase Class Variable	Variables que están almacenadas en la clase y cuyo valor se comparte por todas las instancias de la clase.
Variable de Instancia Instance Variable	Variables que toman valores diferentes en cada instancia u objeto.
Variable de Estado State Variable	Termino alternativo a Variables Clase e Instancia
Método Method	Un método es un procedimiento que realiza un servicio.
Encapsulación Encapsulation	El acceso a la estructura de datos del tipo abstracto de datos se realiza utilizando los métodos en él definidos.
Enlace Dinámico Dynamic Binding, Late Binding	La elección de la implementación a ejecutar se selecciona en tiempo de ejecución.
Enlace Estático Static Binding, Early Binding	La elección de la implementación a ejecutar se selecciona en tiempo de compilación.
Mensaje Message	Solicitud de un servicio
Polimorfismo Polymorphism	El mismo mensaje puede realizar diferentes acciones dependiendo del objeto que lo reciba.
Protocolo Protocol	Conjunto de mensajes a los cuales un objeto puede responder.
Interfaz Interface	El interfaz es el conjunto de métodos definidos para las instancias de la clase.
Herencia Inheritance	Aplicar información general a la situación actual concreta.

Tabla 2.2. Resumen de la Terminología de Objetos.

2.4 MARCOS versus OBJETOS

Aunque los Sistemas Basados en el Conocimiento formalizados en Marcos y los Sistemas Orientados a Objetos producen *Modelos Intermedios* que acercan el Modelo Interno que utiliza la computadora al Mundo Externo, las **motivaciones** que tenían la Inteligencia Artificial y la Ingeniería del Software al crear los Marcos y los Objetos eran diferentes. Mientras que la Inteligencia Artificial, concretamente la Representación del Conocimiento, tenía como motivación encontrar esquemas de representación que fueran capaces de modelizar el conocimiento y el razonamiento que utilizan expertos humanos en la resolución de problemas complejos basados en la experiencia, la Ingeniería del Software buscaba nuevas técnicas que, modelizando problemas no basados en el conocimiento, permitieran construir software de forma más: fiable, barata, fácil de utilizar, que fuera modular, reutilizable y mantenible. Sin embargo, para llevar a cabo objetivos tan dispares, aunque no contrapuestos, ambas utilizaron una arquitectura basada en la estructuración jerárquica de los datos que intervienen en la resolución del problema.

En la década de los setenta, se desarrollaron las primeras herramientas que implementaban Sistemas Basados en Marcos y los primeros lenguajes de Programación Orientados a Objetos. Algunas de estas herramientas basadas en Marcos se implementaron utilizando POO y, de esta forma, las herramientas comenzaron a proporcionar a sus usuarios conceptos que, aunque no formaban parte del formalismo de Marcos, sí añadían nuevas funcionalidades a éste. Por este motivo, se empezaron a importar al formalismo de Marcos, concretamente a las herramientas que implementaban los Marcos, conceptos de la POO.

Durante los últimos años, dada la similitud, al menos superficial, que existe entre el Paradigma de Representación del Conocimiento basado en Marcos y el Paradigma de la Orientación a Objetos, desde varios frentes se ha comenzado a estudiar las relaciones entre la IA y la OO. Prueba de ello son, por ejemplo, la serie de artículos que se han escrito en la revista *IEEE Expert* sobre *Programación Orientada a Objetos en Inteligencia Artificial,* serie que comenzó en Diciembre de 1990 con la finalidad de mostrar los tipos de problemas de IA para los cuales la POO proporciona una solución computacional eficiente [Alpert et al., 90]; o, preguntas planteadas en el panel *OOP and AI* [Ibrahim et al., 91] en la OOPSLA'91 como: ¿Qué puede proporcionar la POO a la IA y

viceversa?, ¿En qué se diferencian?, ¿Puede condicionar una el futuro de la otra?, ¿Cuál de ellas?, ¿Es posible la integración parcial de estas áreas?, ¿Son los objetos deseables para todas las tareas de IA?, ¿Para cuáles no?, ¿Que les falta a las jerarquías de clases de la OO que las evitan las aplicaciones de la IA?, ¿Qué proporciona los lenguajes OO a la IA que Lisp no proporciona?, ¿Cómo puede la POO soportar herramientas ventajosas en el diseño de Sistemas Basados en el Conocimiento?; o, preguntas planteadas en medios de comunicación informales, pero medio habitual de comunicación en la comunidad científica, como son las *News*. Las preguntas [De Francesco, 93], planteadas allí en Marzo de 1993, fueron las siguientes: ¿Se puede decir que existe un marco teórico para estructuras jerárquicas, de la cual, los Objetos y los Marcos son dos implementaciones distintas? y, ¿Está justificada la unión de los Marcos y los Objetos con el fin de obtener un paradigma unificado?.

Las respuestas a estas preguntas son variopintas. En ocasiones, existe un acuerdo tácito en las respuestas y, en otras, las respuestas llegan a ser contradictorias. Las opiniones que diversos autores dan se han agrupado en las siguientes áreas:

a) Motivaciones

Básicamente, los autores consultados coinciden en que las motivaciones de ambas áreas son diferentes. Para Shrobe [Ibrahim et al., 91], la POO es una técnica de programación que se viene utilizando en IA desde finales de los años setenta, que tiene unos intereses distintos a los que tiene la propia IA. Pues mientras a la IA le interesa cómo los expertos conciben el mundo, cómo organizan sus esfuerzos para resolver los problemas, qué abstracciones emplean en la modelización de problemas, a la POO le interesa proporcionar medios de programación ampliables y de gran alcance.

Para Snyder [Snyder, 89] el objetivo de la Representación del Conocimiento es representar el *conocimiento* de tal forma que soporte procesos de razonamiento complejos, frente al de la POO que es representar la *conducta* del sistema.

b) Terminología.

Ballim [De Francesco, 93] opina que la razón por la que los Marcos y los Objetos son dos conceptos distintos se fundamenta en que la terminología que ambos utilizan es distinta.

c) Sintaxis

La sintaxis es una diferencia importante entre Marcos y Objetos [De Francesco, 93]. Los Marcos son más fáciles de escribir que los Objetos, cuya sintaxis depende del lenguaje subyacente y del gran número de características de los sistemas orientados a objetos. Sin embargo, para Hubscher [De Francesco, 93], la sintaxis no es una diferencia importante y depende del sistema que se está utilizando.

La realidad es que existen muchas herramientas y lenguajes que permiten implementar Marcos. Cada uno de ellos, al igual que ocurre con los distintos lenguajes de POO, tiene una sintaxis determinada. Dada la gran variedad de lenguajes que implementan estos conceptos, la comparación se deberá hacer a nivel conceptual. Es decir, comparar el formalismo de Marcos, no las implementaciones de los Marcos, con el Diseño Orientado a Objetos, en vez de compararlos con lenguajes de Programación Orientados a Objetos. Bajo este enfoque, la sintaxis no es importante.

d) Funcionalidad

Para Hubscher [De Francesco, 93], las funciones que los Marcos y los Objetos realizan es la misma, sin embargo, algunas "implementaciones" se realizan de manera más natural con Marcos que con Objetos. La elección de Marcos u Objetos se basa en diferentes estilos de programación. No obstante, De Francesco [De Francesco, 93] opina que, en la mayoría de los casos, se puede alcanzar la misma funcionalidad con Marcos y con Objetos. La única excepción es la equiparación, que no se puede realizar en la OO.

Para Ibrahim [Ibrahim et al., 91], los avances que se han producido en IA no hubieran sido posible sin la POO. Con todo, existen temas como el de clasificación, en el cual, la POO es inadecuada. El problema se encuentra en que la POO crea las clases antes de crear sus instancias y, por consiguiente, no se puede implementar sistemas en los que, a partir de las instancias, se construyen las clases.

Para Perrot [Ibrahim et al., 91], el papel de los objetos en la IA es doble. Por un lado, es una herramienta de programación y, por otro, es una base para la Representación del Conocimiento. Estos dos aspectos se basan en el hecho de que el mecanismo de clases/instancia de la POO proporciona una implementación eficiente de los Conceptos/Instancias del mundo real.

Para Schneider [De Francesco, 93], la diferencia esencial entre la OO y los lenguajes de Marcos no reside en los objetos o en las diferencias entre sus estructuras, pero sí en el hecho de que un lenguaje de Marcos proporciona una gran cantidad de herramientas para generar una Base de Hechos. Para este autor, un lenguaje de Marcos es una extensión de un lenguaje Orientado a Objetos.

e) Herencia

En los Marcos la herencia es dinámica [De Francesco, 93], es decir, los valores de las ranuras de los marcos de niveles superiores no son propagados hacia las instancias, pues desde las instancias se busca en la jerarquía el marco que contiene la propiedad buscada. Sin embargo, en los objetos, se rellenan las instancias en cuanto son creadas. A esto, Hubscher [De Francesco, 93] añade que en los Marcos se tienen jerarquías basadas en la estructura y construidas por especialización o inclusión, y en los sistemas orientados a objetos se tienen jerarquías basadas en conductas similares. Por tanto, en IA se heredan estructuras y en OO se heredan conductas.

f) Encapsulación

En este punto todos los autores están de acuerdo: los Marcos, a diferencia de los objetos, no están encapsulados.

g) Demonios y Métodos

Conceptualmente, los demonios de los Marcos son similares a los métodos de los Objetos. Los demonios se utilizan para computar valores de varias ranuras o para mantener la integridad de la Base de Conocimientos cuando una acción se realiza en un marco y afecta a otros. Sin embargo, los métodos de OO son más generales pues proporcionan un tipo general de computación y soportan encapsulación y polimorfismo [González et al. 93].

2.5 CONSECUENCIAS DEL ESTADO DE LA CUESTION

El estudio realizado en el Estado de la Cuestión lleva a concluir que una clave de la Representación del Conocimiento se encuentra en introducir en la computadora *Modelos Internos* que sean lo más parecido posible al conocimiento, y al uso que de él se

hace en la realidad. Sólo entonces, las respuestas dadas por los sistemas inteligentes se asemejan a las respuestas dadas por los expertos en el Mundo Externo. La adecuación entre el Modelo Interno y la realidad depende del tipo de conocimiento presente en el dominio, del Formalismo de Representación del Conocimiento elegido para construir la BC, y de sus limitaciones en la representación y en las inferencias. Dado que actualmente no existe un formalismo de representación que presente todas las características deseables expuestas por Davis [Davis et al., 93] y por Rich [Rich et al., 91], el Formalismo de Marcos se ha convertido en el paradigma más utilizado en la representación del conocimiento si el conocimiento del dominio está organizado en base a conceptos y, la orientación a objetos, en el paradigma de programación que mejor implementa las estructuras taxonómicas de Representación del Conocimiento [Woods, 91].

Se han percibido las siguientes deficiencias en los trabajos realizados hasta hoy en el formalismo de Marcos: deficiencias en la representación que pueden originar, o no, deficiencias en las inferencias.

1. No existe una teoría aceptada en la comunidad científica sobre los elementos que deben aparecer en el paradigma de Marcos. En el apartado 2.2, se ha mostrado cómo todos los autores están de acuerdo en un núcleo de conceptos básicos, aunque, en ocasiones, la terminología utilizada para nombrarlos sea diferente. Otros conceptos, como el concepto Metaclase, Prototipos o Referencia Autocontenida, son utilizados por muy pocos autores, lo cual hace dudar de su eficacia a la hora de utilizarlos en la formalización de BB.CC.

   Consecuencia inmediata de la ausencia de una teoría, es la gran libertad y carencia de pautas que debe seguir el IC al formalizar. Normalmente, son los entornos que implementan marcos los que restringen la formalización realizada por el IC y los que determinan si el conocimiento está sintácticamente bien o mal formalizado. Esto lleva al IC a formalizar pensando más en el lenguaje del entorno y en las capacidades que éste le permite, que en formalizar, en sentido purista, de acuerdo a las normas de un formalismo, normas que por otro lado no existen.

   Todo ello lleva a afirmar que, actualmente, existe un desfase teórico entre los entornos que implementan el conocimiento y el formalismo que lo representa. Por tanto, se hace necesario crear una base teórica en la

b) Si los dos operandos son tipos básicos definidos en la jerarquía de tipos, entonces, se pasa a ejecutar físicamente la operación en la computadora y se almacena el resultado de la operación en la ranura *Resultado* del marco instanciado.

Supóngase que se desea ejecutar el procedimiento Y(Operando1, Operando2, Resultado) instanciando los parámetros *Operando1* y *Operando2* con los valores almacenados en los parámetros del mensaje. Una vez instanciados los parámetros, el procedimiento Y, definido en el marco clase, envía un mensaje al objeto representado por el *Operando1* en la jerarquía de tipos básicos, con el operando Y y con el *Operando2* como parámetros. Es entonces, cuando llega el mensaje al objeto asociado al *Operador1*, cuando se ejecuta físicamente la operación en la computadora.

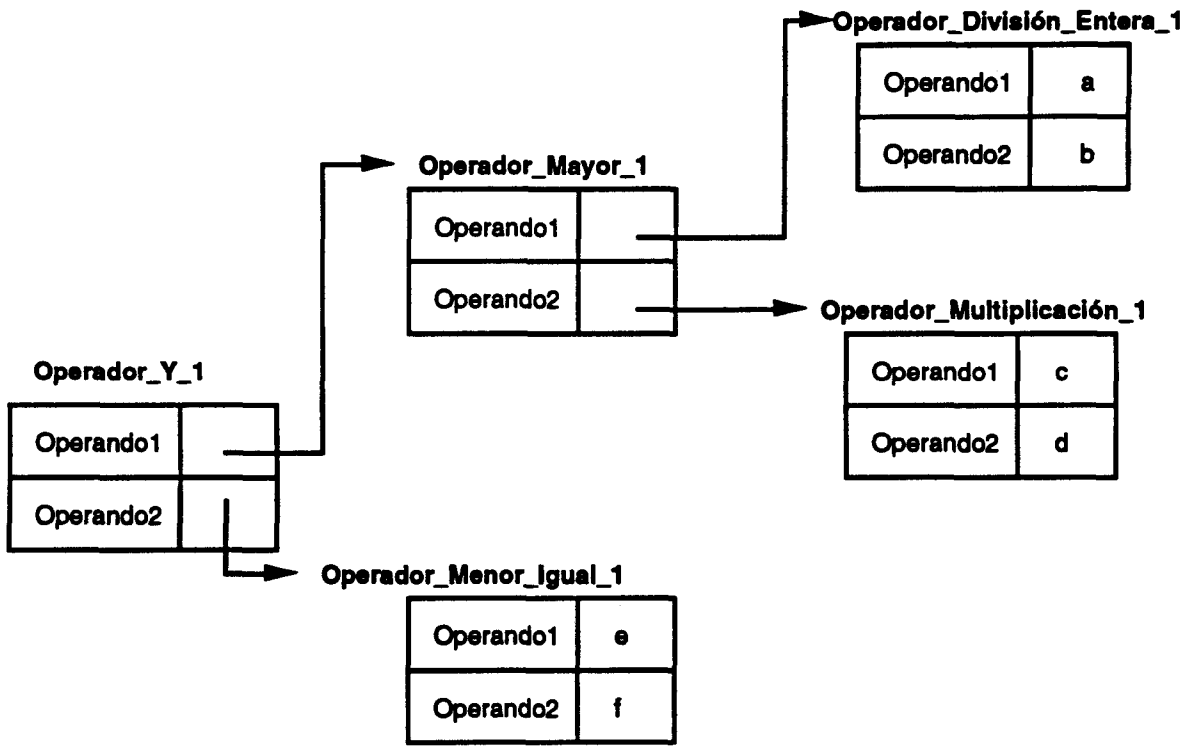


Figura II.6. Ejemplo de una Operación Compuesta

Supóngase que se desea representar en la *Jerarquía de Operaciones*, la expresión lógica siguiente: "[(a Div b) > (c*d)] Y [e=<f]". En la figura II.6 se muestran los marcos instanciados asociados a esta expresión, donde *Operador_Y_1*, *Operador_Mayor_1*, *Operador_División_Entera_1*, *Operador Multiplicación_1* y *Operador_Menor_Igual_1* son instancias de los marcos *Operador_Y*, *Operador_Mayor*,

que se definan claramente todos y cada uno de los conceptos que intervienen en el paradigma y sus inferencias asociadas.

2. A diferencia de lo que ocurre en el Paradigma de la Orientación a Objetos, no existen en el Paradigma de Marcos criterios de diseño que ayuden al IC a formalizar BB.CC. en Marcos, es decir, a seleccionar los marcos y las propiedades que deben aparecer en cada uno de ellos. Tampoco existe un conjunto definido de restricciones que impidan la definición, el uso y la asociación de conceptos que no se corresponden con la realidad, y que sí puede introducir el IC al formalizar.

3. El IC apenas incluye conocimiento sobre relaciones "ad hoc" en una BC, y si las incluye, dichas relaciones no se utilizan para realizar inferencias. La razón, es que las relaciones a medida no poseen unos procedimientos generales en el paradigma. Por este motivo, estas relaciones se suelen utilizar como meras propiedades.

4. Apenas existe bibliografía sobre las técnicas de equiparación en Marcos. El tratamiento de la estructura compacta de la información y la distribución de las propiedades en la jerarquía favorecen la existencia de propiedades compartidas y evitan la presencia de conocimiento redundante en ella. Esto hace que la estructura a equiparar sea compleja, pues, aunque la equiparación se realiza con uno o varios marco(s) de la BC, hay que tener en cuenta toda la información almacenada en todos los marcos accesibles desde ellos.

5. El razonamiento con conocimiento incierto en un SBM es a medida. La incertidumbre se puede introducir en el SBM al construir la BC o al razonar con el conocimiento en ella almacenado. Se hace necesario representar la incertidumbre en los marcos, y crear nuevos procedimientos y, o, modificar procedimientos ya existentes que trabajen con el conocimiento incierto y realicen inferencias correctas.

6. Los algoritmos de herencia basados en la distancia "Inferencial" en grafos acíclicos dirigidos detectan la presencia de ambigüedades, pero no las resuelven. Es necesario adaptar este concepto al formalismo de Marcos y resolver las ambigüedades.

Las deficiencias en las inferencias y las limitaciones en la expresividad encontradas en el paradigma de Marcos ocasionan importantes pérdidas de conocimiento. Sin embargo, aunque ello origina diferencias en las respuestas dadas por el humano y por el sistema inteligente, el principal problema es la ausencia de Modelos Formales basados en Marcos, construidos desde una orientación al usuario, que permitan representar y razonar, entre otros, con el conocimiento de sentido común y con el conocimiento incierto presente en el dominio de la aplicación. Se plantea pues, la necesidad de crear Modelos independientes del dominio construidos desde esta perspectiva, modelos que, por otro lado, no existen actualmente en el mundo.

3. PLANTEAMIENTO

3. PLANTEAMIENTO

Como ya se vió en el Estado de la Cuestión, los formalismos de representación del conocimiento deben crear Bases de Conocimientos que sean sustitutos fieles de la realidad en la computadora, permitiendo la representación de los aspectos relevantes y obviando aquellos que no lo son. Además, las inferencias realizadas sobre el conocimiento de la BC no sólo deben ser correctas en el dominio dado, sino que, además, deben ser computacionalmente eficientes. Los formalismos de representación del conocimiento que posean estas características, crearán *Modelos Internos* en la computadora más próximos a la realidad que aquellos formalismos que no las posean. El problema, es la ausencia de Modelos que ayuden al IC a formalizar cuando utiliza técnicas concretas de representación del conocimiento.

En base a lo anteriormente expuesto, esta Tesis tiene como finalidad crear un **Modelo de Diseño Orientado a Marcos** que, situado en el Nivel Simbólico e intermedio entre el *Modelo Interno* y el *Modelo Cognoscitivo*, disminuya las pérdidas de conocimiento que se producen al formalizar la realidad en BB.CC. El **Modelo de Diseño Orientado a Marcos** será un puente que aproximará la realidad al *Modelo Interno*, creando un *Modelo Formalizado* de la realidad que disminuirá las pérdidas de conocimiento, que si bien no ocurren simultáneamente al construir SS.BB.CC., si coexisten en él.

a) Se acercará el *Modelo Formalizado* al *Modelo Cognoscitivo* y a la realidad al capturar automáticamente, mediante un conjunto de inferencias definidas en el Modelo que se ejecutan al construir la BC del sistema, parte del conocimiento presente en el Nivel de Conocimiento, conocimiento que, por otro lado, el IC no formaliza en su totalidad cuando construye BB.CC. Además, también se acercará el *Modelo Formalizado* a la realidad al incluir en el formalismo nuevos conceptos que incrementan su expresividad, la claridad y la sencillez de la notación, sin complicarla; al modificar las técnicas de inferencia permitidas en él; y, al incluir otras nuevas, como las donaciones.

b) Por otro lado, se acercará el *Modelo Interno* al *Modelo Formalizado* al implementarlo en la computadora utilizando el paradigma de la orientación a objetos.

Para conseguir este Modelo de Diseño Orientado a Marcos, se plantean las siguientes actividades:

a) Dada la estrecha relación entre los formalismos de representación del conocimiento y sus técnicas de inferencia asociadas, en el Modelo de Diseño se definirán dos dimensiones (apartado 5.1): una dimensión para el conocimiento y otra dimensión para el razonamiento.

b) Una sistematización del concepto de Marco y de los elementos básicos que lo forman. Para ello, se definirá, de forma exhaustiva, el vocabulario, la sintaxis y la semántica subyacente a cada uno de estos conceptos (apartado 5.2). Esto determinará, en la dimensión del conocimiento, el conjunto de elementos permitidos en los marcos, sus relaciones, y el conjunto de inferencias que se deben realizar automáticamente al construir el *Modelo Formalizado* de cualquier aplicación, inferencias que introducen en la BC del sistema nuevo conocimiento que el IC normalmente no formaliza.

c) Se aproxima el formalismo a la realidad al incrementar su expresividad. Así, se introducirán en el formalismo conceptos nuevos, como el concepto *Representatividad*, que permitirá al IC representar conocimiento de la realidad que antes no podía representar en el *Modelo Formalizado*, y que, sin embargo, sí poseía el experto (apartado 5.2.2.2.1).

d) Se relaciona el formalismo con la realidad al introducir en el formalismo técnicas de equiparación que trabajan con el conocimiento incierto del dominio. La técnica de equiparación que en este trabajo se propone (apartado 5.3.1), combina la certeza de las propiedades conocidas en una entidad por el usuario de la aplicación, y la representatividad de las propiedades asociadas a los marcos clase dada por el experto y definida por el IC al construir el *Modelo Formalizado* de la aplicación.

e) Se permitirá la definición y el razonamiento con relaciones "ad hoc". Se acerca el formalismo a la realidad al permitir la definición de relaciones "ad hoc" en el *Modelo Formalizado* (apartado 5.2.2.1.2) y al crear una nueva técnica de inferencia llamada *Donación* (apartado

5.3.2.4) que trabaja con estas relaciones. Esta nueva técnica, se basa en la cesión de propiedades (apartado 5.3.2) y, combinada (apartado 5.3.2.2) con la técnica de cesión clásica llamada Herencia (apartado 5.3.2.3), incrementará sustancialmente la compartición o cesión de propiedades en el *Modelo Formalizado*.

f) Se construye un entorno (apartado 5.4) que permitirá al IC formalizar automáticamente y validar la BC sin necesidad de implementarla, evitando la creación o adaptación del *Modelo Formalizado* a un *Modelo Interno*.

Para llevar a cabo todas estas actividades, se ha subdividido este trabajo en el conjunto de fases que, a continuación, se citan y se describen brevemente:

1. Dos Dimensiones del Modelo de Diseño. (Apartado 5.1)

En este apartado se definen dos dimensiones del Modelo de Diseño Orientado a Marcos: la dimensión del conocimiento y la dimensión de las inferencias o del razonamiento.

2. Conocimiento: Teoría de Marcos. (Apartado 5.2)

Se crea una base teórica que uniformiza el concepto de Marco y establece un conjunto de requisitos sintácticos y semánticos que impedirán la definición de elementos no permitidos y el uso indebido de ellos. Por ello, se ha definido con rigurosidad el vocabulario, la sintaxis y la semántica que intervienen en el concepto de Marco, sistematizando así dicho concepto y la terminología empleada.

2.1 Marcos. (Apartado 5.2.1)

Se definen los tipos de Marcos que pueden existir en el *Modelo Formalizado* la BC de un SBM.

2.2 Ranuras. (Apartado 5.2.2)

Se definen los requisitos sintácticos y semánticos que deben cumplir las relaciones que conectan Marcos en el *Modelo Formalizado* y las inferencias que se deben realizar en el Modelo de Diseño al crearlas.

En este apartado, también se definen los tipos de propiedades permitidas en el Modelo de Diseño y se introduce el concepto de *Representatividad* de una propiedad.

2.3 Sintaxis. (Apartado 5.2.3)

Se introduce una notación tabular que almacena el conocimiento expresado en los apartados anteriores y que facilita al IC la descripción de los marcos que forman el SBM.

3. Razonamiento: Inferencias en Marcos. (Apartado 5.3)

En este apartado se modifican algunas de las técnicas de inferencia más utilizadas en los Sistemas Basados en Marcos, permitiendo que estas técnicas trabajen con las nuevas incorporaciones realizadas en el formalismo. Al mismo tiempo, se crean nuevas técnicas de inferencia que incrementan la eficacia del formalismo al aproximar las respuestas de la computadora a las proporcionadas por el ser humano.

3.1 Equiparación. (Apartado 5.3.1)

Se desarrolla una técnica de equiparación que trabaja con el conocimiento incierto presente en el dominio y con la *Representatividad* de las propiedades de los conceptos del dominio.

3.2 Cesión de Propiedades. (Apartado 5.3.2)

Se define una nueva técnica de cesión de propiedades llamada *Donación* que utiliza las relaciones "ad hoc" definidas por el IC para realizar nuevas inferencias. Igualmente, se modifica la cesión de propiedades basada en Herencia al adaptar conceptos de herencia múltiple en grafos

acíclicos dirigidos bipolares homogéneos, no monótonos a Marcos. Ambas, Herencia y Donación, se combinan en un procedimiento que incrementa poderosamente la compartición y distribución de propiedades en la jerarquía de Marcos.

4. Diseño del Modelo Orientado a Marcos. (Apartado 5.4)

Se ha construido un entorno que guía al IC en la creación del *Modelo Formalizado* de un SBC. Se trata de una cierta recursividad. Es decir, el entorno es un SBM que almacena todo el conocimiento descrito en el apartado 5.2 y proporciona todas las inferencias en ella permitidas y descritas en el apartado 5.3. Este entorno, o SBM, ayudará al IC a formalizar cualquier SBM.

4. HIPOTESIS DE TRABAJO

4. HIPOTESIS DE TRABAJO

1. Se supone el conocimiento del Modelo Cognoscitivo validado.

En este trabajo se supone que el conocimiento que el IC formalizará en un Modelo Formalizado, que tiene como base el Modelo de Diesño Orientado a Marcos que en este trabajo se propone, ha sido validado por el IC antes de comenzar la etapa de formalización. El objetivo es asegurar que si existen diferentes respuestas, éstas no se deben a la existencia de conocimiento erróneo en la BC.

2. Suposición del Mundo Cerrado [Reiter, 78].

Se supone la BC completa respecto a las propiedades en ella descritas. Cualquier conocimiento que no se encuentre almacenado en la BC, por omisión, se considera que no es relevante en el dominio y, por consiguiente, es falso. Con ella se asegura que si existen lagunas de conocimiento en la BC, dicho conocimiento no es importante en la aplicación.

3. Se trabajan con valores discretos de las propiedades.

Dado que un SBM está formalizando la realidad, y que en ésta existen valores contínuos, en teoría, se podrían definir los valores de las ranuras como funciones contínuas. Sin embargo, esto significaría complicar tremendamente el motor de inferencia a la hora de realizar la equiparación de las entidades con los marcos clase de la BC y posteriormente aplicar herencia. Esta complicación no se traduciría en un espectacular acercamiento entre el mundo real y el sistema formalizado, dado que en el estado actual, siempre se puede simular una función contínua mediante un conjunto suficientemente grande de valores discretos.

4. No se expresa el conocimiento procedimental del dominio utilizando reglas.

Se utilizarán procedimientos, exclusivamente, por dos motivos:

1. En primer lugar, la inclusión de reglas en el Modelo que en este trabajo se propone, transformaría el Modelo de Diseño Basado en Marcos en un modelo híbrido de representación del conocimiento que tendría unas técnicas de inferencia y control que trabajarían simultáneamente con

ambas representaciones. La inclusión de reglas distorsionaría el modelo orientado a marcos. Por este motivo, no se han introducido.

2. En segundo lugar, las reglas definidas en las facetas procedimentales de las propiedades tampoco se han considerado porque habría que construir un conjunto de procedimientos que continuamente estarían "chequeando" la regla para ver si ésta es aplicable. En este caso, la expresión del conocimiento mediante procedimientos, evita este "chequeo" continuo al permanecer los procedimientos inactivos o latentes en el marco hasta que fuera solicitada su ejecución, sólo entonces son ejecutados.

5. No se utiliza el conocimiento procedimental en los valores activos y métodos.

Se deja como futuros trabajos la elaboración de un procedimiento independiente del dominio que utilice los valores activos y los métodos para realizar inferencias con el conocimiento procedimental expresado en la BC.

5. RESOLUCION

5.1. DOS DIMENSIONES DEL MODELO DE DISEÑO.

Dada la estrecha relación existente entre las técnicas de representación y las técnicas de inferencia que trabajan con ellas, y dado que no se puede hablar de *conocimiento sin hablar de razonamiento ni viceversa*, el desarrollo de un Modelo de Diseño pasa por tratar estas dos dimensiones. Por ello, en este trabajo se trata la dimensión del conocimiento y la dimensión del razonamiento como los únicos pilares sobre los que se apoya el Modelo de Diseño Orientado a Marcos.

El Modelo de Diseño que en este trabajo se propone tiene como meta crear una base teórica sobre la cual se puedan construir modelos formalizados en Marcos. La idea es la siguiente: si se crea un Modelo de Diseño para el paradigma de Marcos y se introducen en dicho Modelo nuevos elementos de representación y nuevas técnicas para realizar inferencias, capturando y razonando con el conocimiento de sentido común y con el conocimiento incierto del dominio, entonces, el Modelo Formalizado así construido estará más cercano a la realidad.

Cada una de las BB.CC. que se construyen utilizando este Modelo de Diseño constituyen un Modelo Formalizado de la realidad. La diferencia entre el Modelo de Diseño y el Modelo Formalizado es la siguiente: mientras que el Modelo de Diseño es único, es independiente del dominio y trata del formalismo de Marcos, el Modelo Formalizado es dependiente del dominio, existirán tantos Modelos Formalizados como BB.CC. se hayan construido, y, trata de la realidad.

Los beneficios de construir un Modelo Formalizado sobre el Modelo de Diseño que en este trabajo se propone, frente a no utilizar dicho Modelo, son los beneficios proporcionados por el Modelo de Diseño. Es decir, representar y razonar con la representatividad de las propiedades, con las relaciones "ad hoc" y con el conocimiento incierto del dominio. A ellos hay que añadir un conjunto de pautas y de restricciones que el Modelo de Diseño lleva asociadas y que, al haberse implementado, se gestionan automáticamente.

Dado que el Modelo de Diseño que en este trabajo se propone es un Modelo Orientado a Marcos, el siguiente esquema ha sido el seguido en la descripción de los elementos que forman el Modelo.

a) Dimensión del Conocimiento

a.1) Marcos

Marcos Clase

Marcos Instanciados

a.2) Ranuras

Relaciones

Relaciones Estándar

Relación SubClase

Relación Instancia

Relaciones No Estándar

Relaciones a Medida

Propiedades: Representatividad

b) Dimensión del Razonamiento

b.1) Equiparación

b.2) Cesión de propiedades

Herencia

Donación

En el apartado 5.2, se muestra una descripción sintáctica, semántica y gráfica, de todos los conceptos que forman la dimensión del conocimiento. En el apartado 5.3, se muestran las técnicas de inferencia que trabajan con ellas, es decir, la dimensión del razonamiento. Una vez que se hayan definido todos estos conceptos, como aplicación práctica, en el apartado 5.4, se muestra el diseño, utilizando Marcos, de un entorno de construcción de SBM construido con Marcos. Se trata de un SBM que gestiona la construcción de aplicaciones concretas basadas en este formalismo. Este SBM ha sido formalizado utilizando el Modelo propuesto en este trabajo.

5.2. CONOCIMIENTO: TEORIA DE MARCOS

5.2.1. MARCOS

Un marco, como ya se vió en el Estado de la Cuestión, es un concepto utilizado en IA para representar conocimiento sobre entidades o individuos, clases o prototipos y eventos existentes en la realidad. Un marco se describe mediante un conjunto de propiedades que definen la entidad, clase o evento que el marco representa. En este trabajo se van a utilizar dos tipos de Marcos: Marcos Clase y Marcos Instanciados. Sin embargo, no se utilizarán marcos MetaClase que representan individuos formados por conjuntos de individuos. En realidad, este concepto viene a expresar, de forma artificial, no natural, y nada eficaz, la relación Pertenece.La notación gráfica que se utiliza para representar los marcos es:

es un Marco Instanciado

es un Marco Clase

5.2.1.1. Marcos Clase

Semánticamente, los Marcos Clase, en adelante MC, representan conceptos, clases y situaciones estereotipadas descritas por un conjunto de ranuras, unas rellenas y otras sin rellenar, comunes a todos los conceptos, clases y situaciones que el MC representa.

5.2.1.2. Marcos Instanciados

Los Marcos Instanciados, en adelante MI, contienen información particular del elemento perteneciente al concepto, clase o situación representado por el marco clase. Estos marcos han rellenado la mayor parte de sus ranuras con los valores específicos del suceso, objeto, entidad o evento que representan y el resto de las ranuras las heredan de los marcos clases de los que son instancia. Por tanto, un marco instanciado

semánticamente va a representar una entidad particular del concepto representado por el marco clase sin rellenar.

5.2.2 RANURAS

Las ranuras que se definen en los Marcos, se agrupan en tres grandes categorías:

a) Las que representan **relaciones** entre Marcos.

b) Las que representan **propiedades**.

c) Las que representan **métodos**.

Se pasa a describir cómo se trata cada una de ellas en esta teoría.

5.2.2.1 Relaciones

Son muchas las relaciones que se definen en un SBM para conectar marcos clase y marcos instanciados. Sin embargo, no todas las relaciones entre estos dos elementos son válidas.

Antes de empezar a analizar las relaciones permitidas entre marcos, se van a explicar las tres notaciones utilizadas en este trabajo:

* Notación de la Teoría de Conjuntos y Diagramas de Venn,

* Notación sobre relaciones, y,

* Notación de Marcos.

Aunque las tres notaciones son formalmente equivalentes, pues todas ellas expresan lo mismo, se utilizará en cada contexto aquella notación que proporcione mayor claridad. Así, si se estudian las relaciones desde el punto de vista de la Teoría de Conjuntos, se utilizará la notación de los Diagramas de Venn. La figura 5.1, muestra los Diagramas de Venn utilizados como notación de la Teoría de Marcos. El universo de discurso en un SBM es la BC del sistema; los marcos clase son conjuntos; y los marcos instanciados son los elementos de los conjuntos.

En la figura 5.1, los conceptos expresados, representados en obscuro, son, de izquierda a derecha: Marco Clase MC; opuesto a Marco Clase MC; inclusión del Marco Clase MC_2 en el Marco Clase MC_1; el Marco Clase MC_1 menos el Marco Clase MC_2; la intersección de los Marcos Clase MC_1 y MC_2; y la unión de ambos marcos.

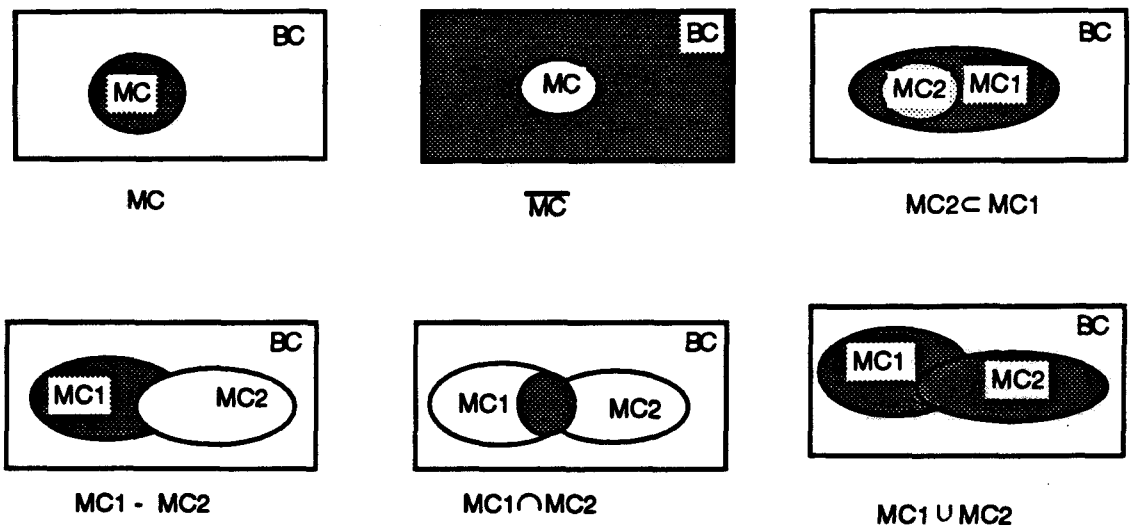


Figura 5.1. Algunas relaciones entre Marcos Clase

Para cualquier relación definida en la Teoría de Marcos, se va a analizar si la relación cumple las propiedades simétrica y transitiva. Demostrar formalmente que las relaciones más utilizadas en los SS.BB.MM., y aquellas que define "ad hoc" el usuario, cumplen las dos, o alguna de las dos propiedades, no es el objetivo de este trabajo. De hecho, el cumplimiento de la propiedad se observa en la mayoría de los casos de manera intuitiva. Las relaciones entre marcos permitirán realizar un conjunto de inferencias, automáticamente, sobre el conocimiento declarativo, explícitamente almacenado en la BC, y crear nuevo conocimiento declarativo mientras se está construyendo la BC, conocimiento que, además de facilitar la labor del IC en la formalización de la BC, podrá ser utilizado, una vez construido el sistema completo, en la realización de inferencias. Las inferencias realizadas sobre la BC, bien al construir el sistema o al utilizarlo, van a depender de la semántica asociada a cada relación y de las propiedades que la relación cumpla.

La segunda notación que se utiliza en este trabajo es la siguiente:

Origen Relación Destino,

donde:

Origen, representa el nombre del marco origen de la relación,

Relación, el nombre de la relación existente entre el origen y el destino, y

Destino, el nombre del marco destino.

perteneciendo, necesariamente, los marcos origen y destino a alguno de los siguientes conjuntos:

$$\mathbf{MC} = \{MC_i \ / \ MC_i \text{ es un Marco Clase}\}$$

$$\mathbf{MI} = \{MI_i \ / \ MI_i \text{ es un Marco Instanciado}\}$$

La tercera notación que se presenta, es la que se utiliza para conectar marcos gráficamente en una jerarquía. En el grafo, las relaciones se representan como arcos unidireccionales etiquetados entre un marco origen y un marco destino. La dirección del arco proporciona la dependencia entre los marcos y, la etiqueta, su semántica. Algunas de las etiquetas más utilizadas son: SubClase, SuperClase, Instancia, Elementos_Clase, Fraternal, Disjunto, No_Disjunto, y "ad hoc".

Se pasa a describir las relaciones más utilizadas en un SBM desde un punto de vista sintáctico y semántico, clasificadas en Relaciones Estándar y en Relaciones No Estándar.

5.2.2.1.1 Relaciones Estándar

En la definición de las relaciones estándar se ha seguido el siguiente esquema:

A.1) Relación SubClase y SuperClase

A.2) Relación Instancia y Elementos_Clase

A.1) Relación SubClase y SuperClase

Gráficamente, estas relaciones se van a representar en una jerarquía de marcos como:

Semánticamente, la **Relación SubClase** entre dos marcos clase, abreviadamente MC_i y MC_j, representa que el marco clase, MC_i, es una clase o especialización de la clase representada por el marco clase MC_j, es decir, el concepto representado por el marco MC_i está incluido o es un subconjunto del concepto representado por el marco MC_j. Los conjuntos de la figura 5.2 representan estos conceptos.

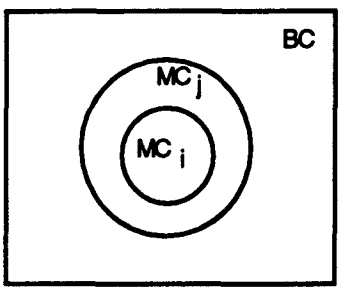


Figura 5.2. Semántica de la Relación SubClase

Sintácticamente la **Relación SubClase** es correcta si se ha definido entre dos marcos clase. La definición de alguna relación clase entre marcos que no sean de este tipo hace a la BC sintácticamente incoherente. Graficamente, la relación se representa como refleja la figura 5.3.

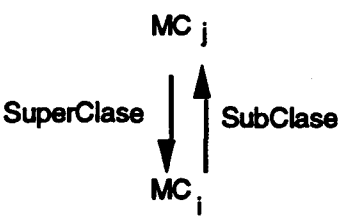


Figura 5.3. Relaciones SubClase y SuperClase

Definición 1: Relación SubClase

$\forall MC_i, MC_j \in MC$:

MC_i *SubClase* $MC_j \Leftrightarrow MC_i \subset MC_j$

Dado que la **Relación SubClase** no es simétrica, es necesario introducir una relación inversa, pero complementaria, que supla esta deficiencia. A la relación inversa de la relación SubClase se la llamará **Relación SuperClase**. Esta relación conectará el marco clase del nivel superior, MC_j, con el marco clase del nivel inmediatamente inferior, MC_i, representando que MC_i es una de las clases en las que se ha especializado la clase o superclase representada por el marco MC_j. Gráficamente, esta relación ya se representó en la figura 5.3. Se pasa a definir formalmente la relación SuperClase.

Definición 2: Relación SuperClase

$$\forall\, MC_i, MC_j \in MC:$$
$$MC_j \; \textit{SuperClase} \; MC_i \Leftrightarrow MC_i \subset MC_j$$

La **Relación SuperClase**, al ser la relación inversa de la relación SubClase, que compensa la ausencia de la propiedad simétrica, deberá crearse de forma automática, nada más introducir una relación SubClase en un SBM. De esta forma, en el Modelo que en este trabajo se propone, se captura automáticamente el conocimiento de sentido común que los IC no suelen formalizar al construir la BC, evitando así la perdida de conocimiento entre el *Modelo Subjetivo* del experto y el *Modelo Formalizado* que el IC está construyendo, y acercando ambos modelos.

Similar a lo que ocurre en la Teoría de Conjuntos, si el marco clase MC_j representa un concepto que se ha especializado en n subconceptos representados por los marcos MC_1, ..., MC_n, entonces, desde MC_j partirán tantas relaciones SuperClase hacia marcos clases MC_1, ..., MC_n, como especializaciones se hayan realizado en dicha clase. Gráficamente, se muestran estas relaciones en la figura 5.4.(a). Por el contrario, si el marco clase MC_j representa un concepto que puede ser clasificado en m conceptos o clases diferentes, entonces, desde MC_j partirán tantas relaciones SubClase como clasificaciones se hayan realizado y, a dicho marco, llegarán tantas relaciones SuperClase como relaciones SubClase se hayan creado. Gráficamente, utilizando la notación de marcos, se han representado en la figura 5.4.(b), estos conceptos.

La transitividad de la relación SubClase proporciona un camino que posibilita la cesión de propiedades de marcos padres a marcos hijos, capturando el conocimiento de sentido común, por el cual, las clases de niveles inferiores poseen el conocimiento

almacenado en las clases de niveles superiores de una jerarquía, o, lo que es lo mismo, las clases de niveles superiores comparten conocimiento con las clases de niveles inferiores.

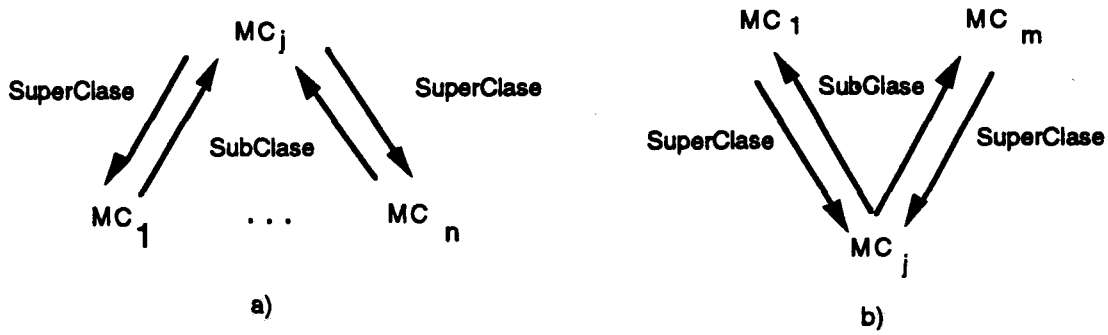


Figura 5.4. n hijos y m padres de un Marco Clase

El problema que presentan las relaciones SubClase y SuperClase en una BC es la presencia de circularidades que hacen a la BC lógicamente incoherente. Formalmente, se define en este trabajo una circularidad entre marcos clase como:

Definición 3: Circularidad de marcos clase.

$\forall MC_k, MC_i \in MC$:

$$\text{Circularidad } (MC_i, MC_k) \Leftrightarrow MC_i \text{ SubClase } MC_k \wedge MC_k \text{ SubClase } MC_i$$

Se produce esta circularidad cuando un marco clase cualquiera, MC_i, es una especialización de otro marco clase MC_k, y, a su vez, dicho marco clase MC_k es una especialización de MC_i. El Modelo que en este trabajo se propone impide la definición de relaciones SubClase entre marcos que presentan circularidades. Gráficamente, se muestra la circularidad en la figura 5.5.

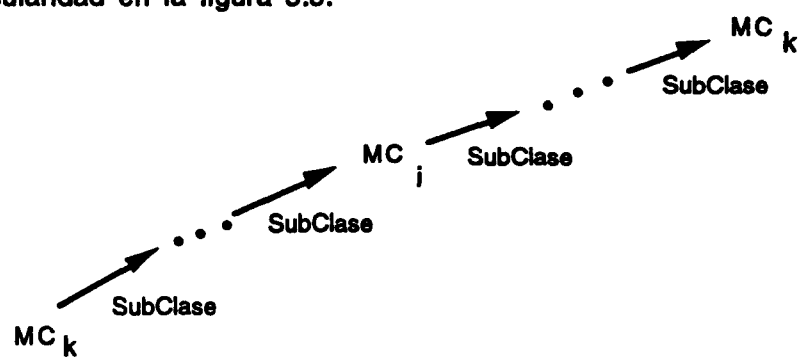


Figura 5.5. Circularidad de Marcos Clases

Por tanto, sintácticamente, una **Relación SubClase**, entre un origen y un destino, está bien definida si:

1. El origen es un marco clase.
2. El destino es un marco clase.
3. No existen circularidades de marcos clase entre el destino y el origen.
4. No se ha definido previamente esta relación entre el origen y el destino.

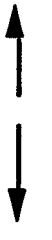
Se puede decir que una **Relación SuperClase**, entre un origen y un destino, está sintácticamente bien definida si:

1. El origen es un marco clase.
2. El destino es un marco clase.
3. No se ha definido previamente esta relación entre el origen y el destino.

Estos prerrequisitos son condiciones que deben satisfacerse antes de crear una relación SubClase o SuperClase en una BC. De este modo, se acerca el *Modelo Formalizado* a la realidad al evitar la presencia de incoherencias en la BC que no existen en la realidad, y al introducir automáticamente conocimiento cierto de la realidad que antes no se introducía.

A.2) Relación Instancia y Elementos_Clase

Gráficamente, estas relaciones se representan en una jerarquía de marcos utilizando la notación:



Instancia — Relación Instancia de una Clase

Elementos_Clase — Relación Elementos_Clase de una Clase

Semánticamente, la **Relación Instancia**, entre un marco instanciado MI_i y un marco clase MC_j, representa que el marco instanciado, MI_i, es un elemento del conjunto representado por el marco clase MC_j, y, por consiguiente, el marco MI_i puede acceder a todas las propiedades definidas en MC_j y en los marcos accesibles desde MC_j. El diagrama de Venn de la figura 5.6, representa el concepto de instanciación en la Teoría de Conjuntos.

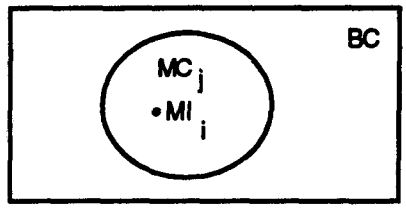


Figura 5.6. Relación Instancia

Sintácticamente, la **Relación Instancia** es correcta si tiene como origen un marco instanciado, MI_i, y como destino un marco clase MC_j. La definición de la relación entre marcos que no se ajusten a estas descripciones hace a la BC sintácticamente incoherente. Utilizando la notación de Marcos, estas relaciones se representan como se muestra en la figura 5.7.

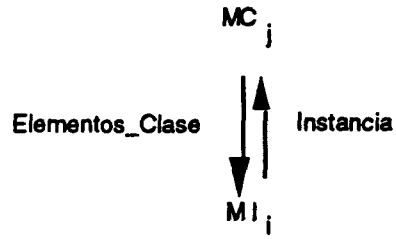


Figura 5.7 Relaciones **Instancia** y **Elementos_Clase**

Formalmente, se define la relación como:

Definición 4: Relación Instancia

$$\forall MI_i \in MI, \forall MC_j \in MC:$$

$$MI_i \textit{ Instancia } MC_j \Leftrightarrow MI_i \in MC_j$$

Al no cumplir la relación Instancia la propiedad simétrica, es necesario dar nombre a su relación inversa con el fin de evitar la pérdida de conocimiento ocasionada por la ausencia de esta propiedad. La **Relación Elementos_Clase**, relación inversa

de la relación Instancia, conectará el marco clase MC_j con el marco instanciado MI_i, representando que uno de los elementos concretos de la clase MC_j es MI_i. Desde MC_j partirán tantas relaciones del tipo Elementos_Clase como elementos o instancias se encuentren en dicha clase y, desde el marco instanciado, MI_i, partirán tantas relaciones del tipo Instancia como a marcos clase pertenezca la instancia, siendo la **Relación Elementos_Clase** sintácticamente correcta si se ha definido entre un marco clase origen, MC_j, y un marco instanciado destino, MI_i. La relación se define formalmente como:

Definición 5: Relación Elementos_Clase

$\forall MI_i \in MI, \forall MC_j \in MC$:

MC_j *Elementos_Clase* $MI_i \Leftrightarrow MI_i \in MC_j$

Pero, si el marco instanciado MI_i es un conjunto formado por uno o varios marcos instanciados: MI_{i1}, ..., MI_{ik}, de tal forma que la relación existente entre los elementos y el marco instanciado es una relación de pertenencia, entonces, será necesario utilizar una nueva relación llamada **Relación Pertenece** que el IC definirá a medida en la BC, y que se estudiará posteriormente en las relaciones no estándares.

Por tanto, una **Relación Instancia**, entre un marco origen y otro destino, está sintácticamente bien definida si:

1. El origen es un marco instanciado.
2. El destino es un marco clase.
3. No se ha definido con anterioridad esta relación entre el origen y el destino.

También lo está la **Relación Elementos_Clase** si:

1. El origen es un marco clase.
2. El destino es un marco instanciado.
3. Anteriormente, nunca se ha definido la relación entre el origen y el destino.

5.2.2.1.2 Relaciones No Estándar

Las relaciones no estándar son relaciones binarias, distintas a las relaciones SubClase e Instancia. El Modelo de Diseño que se propone, distingue dos tipos de relaciones no estándar. Unas, dadas por el paradigma de Marcos, y otras, definidas por el IC cuando construye el Modelo Formalizado del SBM.

Ambos tipos de relaciones deben incorporarse al Modelo por dos motivos. El primero, porque capturan conocimiento de sentido común que el IC normalmente no formaliza por no saber si se puede implementar dicho conocimiento en la herramienta seleccionada y, en segundo lugar, porque si decide implementar el sistema a medida con un lenguaje de alto nivel, orientado a objetos o no, no existen unos procedimientos independientes del dominio que permitan realizar inferencias con este tipo de relaciones. Por tanto, en cualquier relación no estándar, habrá que distinguir la declaración de la relación del uso de dicha relación para realizar inferencias.

Desde el punto de vista de la representación, las relaciones no estándares se utilizan para representar conocimiento adicional sobre la estructura o topología de los marcos que forman el SBM. En este caso, las relaciones deben formar parte del Modelo de Diseño. Pero, la relación también la puede definir el IC para representar dependencias entre conceptos y, en este caso, el Modelo de Diseño debe permitir su definición. Como ejemplo de relaciones estructurales dadas en el Modelo de Diseño, cabe destacar: la relación Fraternal, la relación Disjunto y la relación No Disjunto y, como ejemplo de relaciones que expresan dependencias entre conceptos, las relaciones: *Pertenece*, *Casado con*, *Conectado a*, etc.

Desde el punto de vista del razonamiento, las relaciones se deben utilizar para realizar inferencias en el dominio en el que han sido declaradas. Tener declarada una relación "ad hoc" y no utilizarla para realizar inferencias obliga a usar la relación como si se tratara de una propiedad más de un marco de la BC. La ausencia de unos procedimientos que trabajen con ellas de forma genérica e independiente del dominio en la equiparación y, o, en la cesión de propiedades, ha originado que estas relaciones apenas sean utilizadas en los SS.BB.MM. Se pierde así una importante fuente de conocimiento que modela el conocimiento de sentido común de un determinado dominio y se incrementa el desfase en las respuestas dadas por la máquina y por el ser humano.

El Modelo de Diseño que se propone permite la definición de relaciones estructurales y de dependencia, y proporciona un conjunto de procedimientos independientes del dominio que trabajan con ellas en la equiparación y en la cesión de propiedades. En este apartado se van a analizar, exclusivamente, las relaciones no estándar en su aspecto declarativo. El uso de la relación se analizará en el capítulo 5.3, al tratar las inferencias. Antes de pasar a analizar las relaciones no estándares dadas por el Modelo de Diseño, se van a definir los elementos que deben estar presentes en cualquier relación a medida definida por el usuario.

A) Elementos de una relación "ad hoc".

El IC, al definir una relación "ad hoc" entre dos marcos clase de una BC, deberá especificar los siguientes elementos:

A.1) el nombre de la relación,

A.2) la sintaxis de la relación,

A.3) la relación inversa o recíproca, y

A.4) la semántica de la relación.

A.1) El nombre de la relación.

El IC dota de un nombre a la relación, que será representativo del significado de la relación.

A.2) Sintaxis de la relación.

Sintácticamente, las relaciones "ad hoc" están bien definidas en el Modelo de Diseño si conectan marcos clase según alguna de las configuraciones mostradas en la figura 5.8:

a) El marco clase origen y destino de la relación es el mismo.

b) El marco clase origen de la relación es distinto al marco clase destino.

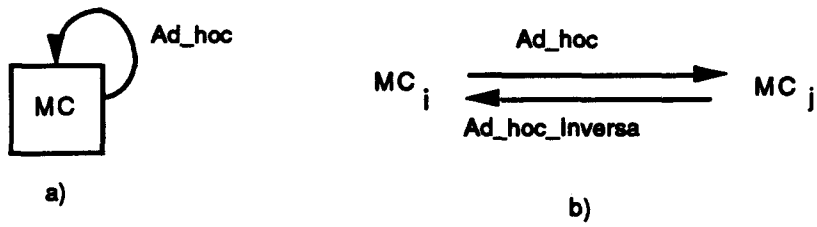


Figura 5.8. Sintaxis de relaciones "ad hoc".

A.3) Relación Recíproca o Inversa.

La definición de una relación "ad hoc" simétrica entre marcos clase obliga a crear una relación recíproca con el mismo nombre al crear la relación entre dichos marcos. Si la relación no es simétrica, el IC deberá nombrar la relación inversa, relación que también se creará automáticamente al introducir la relación en la BC.

A.4) Semántica de la relación.

La semántica de una relación a medida la define el IC especificando un conjunto de *propiedades exportables* o propiedades cedidas en la relación desde el marco clase destino hacia el marco clase origen de la relación. El IC debe seleccionar, entre todas las propiedades definidas en el marco destino, y en sus ascendientes, aquellas propiedades cuyos valores puedan ser exportados o cedidos al marco origen de la relación.

Junto con las propiedades cedidas definidas por el IC en la relación "ad hoc", si la relación es transitiva, la propia relación también se considera como una propiedad exportable. El grado en el cual una propiedad se comparte en una relación "ad hoc" depende de las restricciones impuestas por el IC en dicha propiedad, restricciones que se especifican utilizando un conjunto de condiciones.

Además, se definirán un conjunto de facilidades que permitirán al IC determinar el alcance de la relación "ad hoc", es decir, el número de arcos en los cuales se va a buscar la propiedad P en la relación "ad hoc".

Con todos estos elementos, el IC puede definir en el Modelo Formalizado que construye sobre el Modelo de Diseño que aquí se propone, cualquier relación estructural entre marcos clase del SBM o cualquier tipo de dependencia entre los conceptos que los marcos representan en la BC. En el Modelo de Diseño, se van a llamar

relaciones estructurales a relaciones a medida que informan sobre la estructura, o topología, de los marcos de cualquier BC. Ejemplos de estas relaciones son: la **Relación Fraternal**, la **Relación Disjunto**, la **Relación No Disjunto**, la **Relación Recubrimiento**, etc. En el modelo de representación y razonamiento que en este trabajo se propone, se van a utilizar las tres primeras. No obstante, la modularidad del Modelo permite que el IC defina otras relaciones y sus inferencias asociadas.

Las relaciones "ad hoc" también se utilizan para representar dependencias concretas, a medida, entre conceptos que aparecen en la realidad. En este caso, el nombre de la relación refleja la dependencia entre los conceptos que ésta relaciona. Por ejemplo, se puede crear una relación *Casado con* entre *Hombres* y *Mujeres* para expresar que un *Hombre* se casa con una *Mujer* y viceversa. Igualmente, se pueden crear otras relaciones, como la relación *Pertenece*, *Conexión*, etc.

Las inferencias realizadas con las relaciones "ad hoc" estructurales y las que representan dependencias son diferentes. En las primeras, el razonamiento se realiza con marcos clase, y, en las segundas, con marcos instanciados. En el capítulo 5.3, se presentan las técnicas de inferencia.

Se pasan a describir las relaciones no estándares utilizadas en el Modelo de Diseño que en este trabajo se propone. Para cada una de ellas se especifican todos los elementos definidos en la descripción de las relaciones "ad hoc" y las técnicas de inferencia en las que se utilizan.

B) Relación Fraternal

La **Relación Fraternal** es una relación estructural que crea y gestiona automáticamente el Modelo de Diseño al detectar dos o más marcos clase con el mismo padre. Esta relación y su relación recíproca se representan gráficamente en un SBM como:

Sintácticamente, dos marcos clase cualesquiera, MC_i y MC_j, son hermanos si tanto MC_i como MC_j se entroncan mediante Relaciones SubClase con el mismo marco progenitor, MC_k. Gráficamente, se ha representado esta relación en la figura 5.9, y, formalmente, se define la Relación Fraternal en la definición 6.

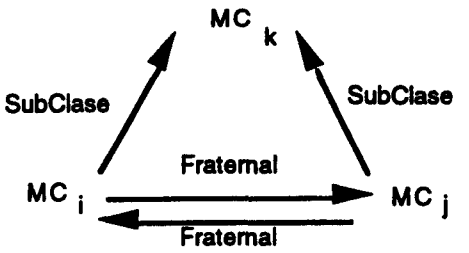


Figura 5.9 Relación Fraternal

Definición 6. Relación Fraternal

$\forall MC_i, MC_j, MC_k \in MC$

$$MC_i \; Fraternal \; MC_j \Leftrightarrow MC_i \; SubClase \; MC_k \wedge MC_j \; SubClase \; MC_k$$

Se observa, intuitivamente, que la Relación Fraternal solamente cumple la propiedad simétrica y carece de la propiedad transitiva. Por ejemplo, las relaciones fraternales de la jerarquía de la figura 5.10, supuestos todos los marcos que en ella aparecen marcos clases, son: *A Fraternal B, B Fraternal A, B Fraternal C, C Fraternal B, C Fraternal D* y *D Fraternal C.* Es evidente, que si se aplica la propiedad transitiva, se llegaría a concluir que existe una relación fraternal entre *A* y *D*, cosa que es completamente falsa.

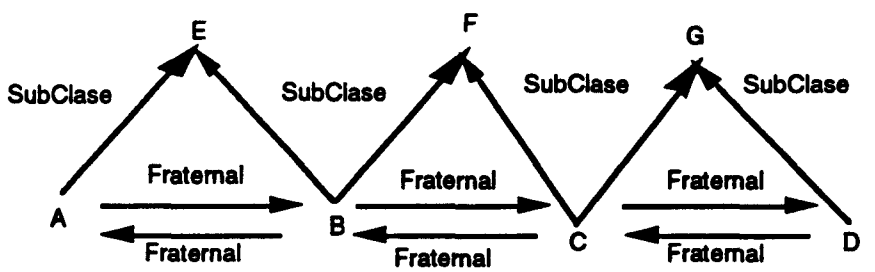


Figura 5.10. Ejemplo de Relaciones Fraternales.

No tiene sentido definir en la relación Fraternal propiedades exportables porque las instancias de una BC nunca van a razonar con esta propiedad. Esta relación únicamente se va a utilizar en el Modelo de Diseño para realizar inferencias en el proceso de equiparación.

Si un marco clase MC_i, es hermano de K marcos, desde MC_i partirán tantas relaciones fraternales como hermanos se hayan definido, y a dicho marco, llegarán

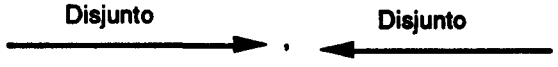
tantas relaciones recíprocas como relaciones directas partan de él. En ambos casos, saldrán K relaciones fraternales y llegarán K relaciones recíprocas fraternales.

Por tanto, la **Relación Fraternal** está, desde un punto de vista sintáctico, bien definida si:

1. El origen y el destino son marcos clase.
2. El origen es diferente al destino.
3. El origen y el destino tienen el mismo padre, y están unidos a él por relaciones SubClase.
4. No se ha definido anteriormente esta relación entre el origen y el destino.

C) Relación Disjunto

La Relación Disjunto es una relación que, explícitamente, define el IC entre dos marcos clase de una BC. Una vez definida la relación entre dos marcos clase, el Modelo de Diseño utilizará esta relación en el proceso de equiparación para realizar inferencias. Gráficamente, esta relación y su recíproca se representan en un SBM como:



Dos marcos son disjuntos, si representan conjuntos disjuntos de entidades. Dos marcos disjuntos nunca pueden instanciarse para producir dos marcos alternativos. Si dos marcos se encuentran relacionados mediante relaciones disjuntas, y dichos marcos se encuentran en niveles distantes de una jerarquía, significa que la jerarquía está desequilibrada, existiendo conocimiento de grano fino en una de sus ramas y conocimiento de grano grueso en otra.

Sintácticamente, la Relación Disjunto está bien definida si dicha relación conecta dos marcos clase diferentes entre los cuales no existe ninguna relación No Disjunto. Aunque la definición pueda parecer trivial, es necesario introducir esta restricción en el Modelo de Diseño para evitar encontrar simultáneamente, en los Modelos Formalizados, relaciones Disjunto y No Disjunto entre dos marcos clase

cualesquiera, o una relación disjunta que tenga como origen y destino el mismo marco clase. Gráficamente, se representa esta relación como se muestra en la figura 5.11.

Figura 5.11. Relación Disjunto

Semánticamente, dos marcos clases MC_i y MC_j son disjuntos, si tanto MC_i como MC_j tienen como intersección el conjunto vacío, es decir, cuando las instancias de MC_i son distintas a las de MC_j, o lo que es lo mismo, los conjuntos o clases representados por ambos marcos no contienen ningún elemento común. Ello obliga al Modelo de Diseño a comprobar que, cuando desde un marco instanciado parten dos o más relaciones Instancias, en los posibles caminos que unen el marco instanciado con el marco raíz de la jerarquía no existen marcos clases para los cuales se haya dicho explícitamente que son disjuntos. Dado que esta relación se va a utilizar en el proceso de equiparación para realizar inferencias, no es necesario definir ningún conjunto de propiedades exportables que proporcionen la semántica.

Formalmente, se definen la relación como sigue:

Definición 7: Relación Disjunto

$$\forall MC_i, MC_j \in MC$$

$$MC_i \textit{ Disjunto } MC_j \Leftrightarrow MC_i \cap MC_j = \phi$$

La relación asi definida, al igual que sucedía con la relación fraternal, cumple la propiedad simétrica, pero no la propiedad transitiva. Por consiguiente, al cumplir la propiedad simétrica, nada más crear la relación directa entre dos marcos clase debe crearse la relación recíproca asociada.

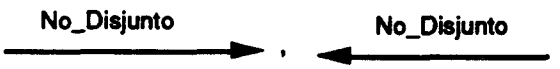
Si un marco clase MC_i es disjunto con k marcos, desde MC_i partirán k relaciones disjuntas, y a dicho marco, llegarán tantas relaciones recíprocas como relaciones directas se hayan definido en él. En ambos casos, k relaciones directas disjuntas y k relaciones recíprocas disjuntas.

La **Relación Disjunto** entre un origen y un destino está sintácticamente bien definida si:

1. El origen y el destino son marcos clases.
2. El origen es distinto al destino.
3. No se ha definido previamente una relación de este tipo entre el origen y el destino.
4. No se ha definido previamente una relación No_Disjunto entre el origen y el destino.

D) Relación No_Disjunto

La Relación No Disjunto es análoga sintácticamente a la relación Disjunto, pero opuesta semánticamente a ella. Gráficamente, la relación No Disjunto y su relación recíproca se representan como:



Dos marcos pueden relacionarse mediante la **Relación No_Disjunto**, si representan conjuntos no disjuntos de entidades. Dos marcos no disjuntos podrán instanciarse produciendo dos marcos alternativos.

Sintácticamente, la Relación No Disjunto está bien definida, si la relación conecta dos marcos clase diferentes que no están unidos por una relación Disjunto. Gráficamente, se muestra esta relación en la figura 5.12.

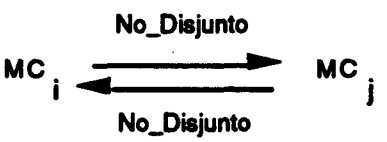


Figura 5.12. Relación No_Disjunto

Semánticamente, dos marcos clases cualesquiera, MC_i y MC_j, son no disjuntos si la intersección de MC_i con MC_j es distinta del conjunto vacío. Es decir, cuando al menos una de las instancias de MC_i es instancia también de MC_j, o lo que es lo mismo,

los conjuntos o clases representados por ambos marcos contienen como mínimo un elemento en común. Formalmente, se define la Relación No_Disjunto, relación que cumple la propiedad simétrica, pero no la propiedad transitiva, como:

Definición 8: Relación No_Disjunto

$\forall MC_i, MC_j \in MC$

MC_i *No_Disjunto* $MC_j \Leftrightarrow MC_i \cap MC_j \neq \phi$

Si un marco clase MC_i, es no disjunto con k marcos, desde MC_i partirán k relaciones no disjuntas, y a dicho marco llegarán k relaciones recíprocas.

La **Relación No_Disjunto** entre un origen y un destino está sintácticamente bien definida si:

1. El origen y el destino son marcos clases.
2. El origen es diferente al destino.
3. No se ha definido previamente una relación de este tipo entre el origen y el destino.
4. No se ha definido previamente una relación Disjunto entre el origen y el destino.

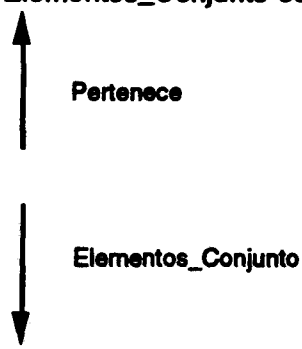
E) Relación Pertenece

La **Relación Pertenece** es una relación que, explícitamente, define el IC entre dos marcos clase de la BC para representar dependencias entre conceptos. Esta relación, se ha incorporado en el Modelo de Diseño que se propone porque tiene sus raíces en el concepto de MetaClase propuesto por Rich y Knight [Rich et al., 91]. Como se vió en el Estado de la Cuestión, ellos definieron las metaclases como clases cuyos elementos (instancias) son a su vez clases o conjuntos. Por ejemplo, el *Real Madrid* es una instancia de los *Equipos de Futbol de Primera División.* A su vez, el *Real Madrid* representa el conjunto de *Jugadores* que juegan en dicho equipo, jugadores que son instancias del concepto *Jugadores en Equipos de Primera División.* Rich y Knight

llaman al marco *Equipos de Futbol de Primera División* marco metaclase; al marco *Real Madrid*, marco instancia o marco clase instanciado de la metaclase; y, a los *Jugadores*, marcos instanciados del marco clase instanciado de la metaclase. Este enfoque, además de ser de difícil comprensión, presenta el problema de no distinguir qué propiedades pueden heredar los *Jugadores* del marco clase instanciado *Real Madrid* y cuales no, pues la relación de instanciación proporciona un camino que posibilita la herencia de propiedades no sujeta a ningún tipo de restricciones.

Con el fin de diferenciar las propiedades heredables de las que no lo son, y poder expresar que determinadas entidades no son instancias sino que pertenecen o son miembros de un conjunto que es una instancia, en este trabajo se propone una nueva relación llamada Pertenece, que conectaría a las instancias con el conjunto instanciado al cual pertenecen.

La definición de la Relación Pertenece se realiza entre el marco clase origen, al cual pertenecen las instancias, y entre el marco clase destino, al cual pertenece el conjunto instanciado. Además, el IC debe especificar el conjunto de propiedades exportables desde el destino hacia el origen, y el número de relaciones Pertenece potenciales que pueden partir de una instancia del marco clase origen de la relación. Por tanto, en el ejemplo anterior, la Relación Pertenece se definiría entre los marcos clase *Jugadores en Equipos de Fútbol de Primera División* y *Equipos de Fútbol de Primera División*, convirtiéndose en una relación a medida, proporcionada por el Modelo de Diseño, y parcialmente definida por el I.C. Desde cualquier marco instanciado, instancia del marco clase *Jugadores de Fútbol en Equipos de Primera División*, podrá partir una relación Pertenece hacia un marco instanciado instancia del marco clase *Equipos de Fútbol de Primera División*. Gráficamente, la Relación Pertenece y su relación inversa Elementos_Conjunto se definen como:



Sintácticamente, la **Relación Pertenece** es correcta si tiene como origen un marco Clase, MC_i, y como destino un marco clase MC_j. La definición de la relación entre marcos que no se ajusten a estas descripciones hace a la BC sintácticamente

incoherente. Gráficamente, esta relación se representa como se muestra en la figura 5.13.

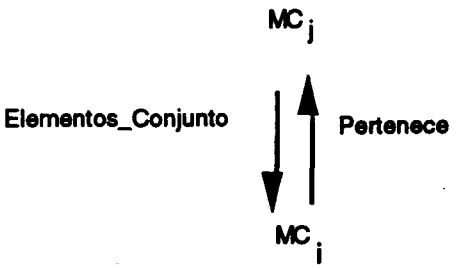


Figura 5.13. Relaciones **Pertenece** y **Elementos_Conjunto**

Formalmente, la nueva relación se define como:

Definición 9: Relación Pertenece

$\forall MC_i, MC_j \in MC$

MC_i *Pertenece* $MC_j \Leftrightarrow MC_i \in MC_j$

La semántica de la Relación Pertenece, la define el IC especificando el conjunto de propiedades exportables en la relación.

Al no ser la Relación Pertenece una relación simétrica, la relación inversa de la Relación Pertenece, **Relación Elementos_Conjunto**, conectará el marco clase MC_j, con el marco clase, MC_i. Desde las instancias de MC_j partirán tantas relaciones de tipo Elementos_Conjunto como elementos contenga dicho conjunto. Igualmente, desde las instancias de MC_i partirán tantas relaciones Pertenece como a conjuntos instanciados diferentes pertenezca la instancia. Por tanto, aunque solamente se defina entre dos marcos clase una única relación pertenece, habrá que especificar la cardinalidad máxima de la relación para, por ejemplo, impedir que un mismo jugador pertenezca a dos equipos de fútbol.

La **Relación Elementos_Conjunto** es sintácticamente correcta, si se ha definido entre un marco clase origen, MC_j, y un marco clase destino, MC_i. Formalmente, se define como:

Definición 10: Relación Elementos_Conjunto

$\forall MC_i, MC_j \in MC$

$$MC_j \textit{ Elementos\_Conjunto } MC_i \Leftrightarrow MC_i \in MC_j$$

Sintácticamente, la **Relación Pertenece** está bien definida si:

1. El origen es un marco clase.
2. El destino es un marco clase.
3. Es la primera vez que se ha definido la relación entre el origen y el destino.

También lo estaría la **Relación Elementos_Conjunto** si:

1. El origen es un marco clase.
2. El destino es un marco clase.
3. Anteriormente, nunca se ha definido la relación entre el origen y el destino.

5.2.2.1.3 Las Relaciones en el Modelo de Diseño

Como conclusiones a este apartado es obligado citar:

1. Los prerrequisitos asociados a cada relación estándar y no estándar son un conjunto de condiciones que se definen en el Modelo de Diseño, condiciones que, por otro lado, deben satisfacerse en cualquier Modelo Formalizado antes de incluir una relación entre marcos de una BC. Con ello, se evita el uso incorrecto de las relaciones y la presencia de incoherencias sintácticas y lógicas en la BC.
2. La gestión automática de las relación inversa o recíproca nada más crear la relación evita pérdidas de conocimiento.

5.2.2.2 Propiedades

5.2.2.2.1. Representatividad de las Propiedades en el Mundo Externo

En este apartado se exponen dos criterios independientes, pero complementarios, que el IC debe tener presente a la hora de definir propiedades en un SBM. Uno de ellos, es el ya conocido criterio de la compartición de propiedades. El otro, es un criterio nuevo que se introduce en el Modelo de Diseño que se propone y al que se le ha llamado *Representatividad.*

A) Compartición de Propiedades

La distribución de las propiedades en los SS.BB.MM. debe favorecer que unos marcos compartan propiedades con otros marcos, evitando la presencia de conocimiento redundante en la jerarquía. Por este motivo, el IC definirá un conjunto de propiedades de Clase y de Instancia en los marcos clase del SBM que caracterizan y discriminan a las instancias de dichos marcos clase. La etiqueta, que da nombre a cada propiedad, refleja la semántica de lo que la propiedad representa, pero no indica la importancia o representatividad de la propiedad en el concepto que el marco representa.

B) *Representatividad* de Propiedades

Además, las propiedades definidas en cada marco deben ser representativas. Es decir, deben proporcionar la suficiente información como para determinar la entidad que el marco está representando, discriminando las entidades representadas por él de las que no lo son. Así, en un mismo marco clase, pueden encontrarse propiedades altamente representativas, entendiendo en este trabajo la *Representatividad* como: especificidad, relevancia, significación, aporte de información o discriminación, de la propiedad en el concepto que el marco representa, y, propiedades generales, meramente descriptivas, informativas y poco o nada discriminativas o representativas. La *Representatividad* depende del dominio en el que se estudien las propiedades.

Por ejemplo, en el marco clase *Perro*, la propiedad *Ladra* es más representativa que la propiedad *Animal Doméstico*, y ésta, más representativa que la

propiedad *Nº de Patas*, pues muchos otros conceptos o marcos clase tendrán también la propiedad *Animal Doméstico*, mientras que ningún otro marco clase tendrá la propiedad *Ladra*.

Se pasa a analizar en mayor profundidad el criterio de *Representatividad* siguiendo el esquema que a continuación se propone:

B.1) Características de la Representatividad.

B.2) Criterios de Representatividad.

B.3) Definición de la Representatividad.

B.4) Cuantificación de la Representatividad.

B.1) Características de la Representatividad

En este apartado, se proponen un conjunto de características que debe tener presente el IC cuando construye un Modelo Formalizado del conocimiento del experto y asigna las representatividades dadas por el experto a las propiedades de un marco de la BC.

- *La Representatividad es un criterio independiente al criterio de Compartición de Propiedades.*

  La distribución y la *Representatividad* de las propiedades son criterios independientes que utiliza el IC a la hora de construir BB.CC. basadas en Marcos.

  Por ejemplo, supóngase el marco raíz de cualquier jerarquía de marcos. Las propiedades almacenadas en dicho marco alcanzan el nivel máximo de compartición, pues cualquier marco de nivel inferior puede consultar el valor en ella almacenado o su definición. Sin embargo, esto no quiere decir que las propiedades almacenadas en el marco raíz tengan una representatividad alta. Por ejemplo, en una BC sobre animales *Vertebrados*, la propiedad *Esqueleto* es absolutamente representativa de la clase, mientras que la propiedad *Nº de Ojos* apenas tiene representatividad, y, ambas son compartidas por todos los marcos de los niveles inferiores.

- *La Representatividad de una propiedad depende del contexto.*

Una BC no está mejor construida que otra porque tenga más propiedades, sino porque las propiedades detalladas sean más representativas, es decir, aporten más información y permitan realizar mejores inferencias con el conocimiento almacenado. Por este motivo, la *Representatividad* de las propiedades definidas en la jerarquía debe proporcionarla el experto en función del uso que se vaya a hacer de ellas en el sistema.

Por ejemplo, en un SBC que aconseje a las personas qué animales domésticos puede tener en casa, la propiedad *Animal Doméstico* es muy representativa. Por el contrario, si el sistema clasifica animales en función del *Número de Patas* que tienen, entonces, la propiedad *Animal Doméstico* no es nada representativa.

- *La Representatividad captura conocimiento de sentido común y evita pérdidas de conocimiento.*

La *Representatividad* captura conocimiento de sentido común y evita pérdidas de conocimiento al establecer un orden en las propiedades de cada marco clase. Este orden debe ser lo más parecido posible al orden en el que las propiedades aportan conocimiento al marco, es decir, discriminan al concepto que el marco representa en la realidad.

Supóngase que se tiene una BC de animales y una entidad E, para la cual se conoce su *Nombre*, su *Peso* y su *Nº de Patas*. Si a un ser humano y a un SBC se les preguntara de qué entidad se trata, a ambos les sería imposible determinar la clase a la cual pertenece la entidad con la información disponible. La causa está en que las propiedades *Nombre*, *Peso* y *Nº de Patas*, apenas tienen representatividad en el dominio de animales en el que se trabaja.

Pero, si se tuviera conocimiento adicional sobre la entidad y se supiera que se trata de un *Animal Doméstico*, entonces, las respuestas dadas por un SBC y por un ser humano podrían ser diferentes. Probablemente, el ser humano respondería que se trata de un *Perro* frente a un *Gato*, o una *Tortuga*, porque, aunque todos tienen el mismo *Nº de Patas*, los *Perros* son los animales domésticos por excelencia. Por contra, el SBM detectaría una ambigüedad al

encontrar la propiedad *Animal Doméstico* definida en todos los marcos anteriormente mencionados.

- *Las propiedades representativas debe ser el criterio utilizado para especializar un marco clase en otros marcos clase.*

  Por ejemplo, supóngase el marco clase *Perro* en el que se definen las propiedades *Ladra* y *Raza*. Ambas propiedades son muy representativas porque conocida la presencia de alguna de ellas en una entidad E, se puede utilizar ésta para equiparar la entidad con el marco clase. Sin embargo, la propiedad *Raza*, al tomar un valor diferente en cada instancia, es más representativa que la propiedad *Ladra*, y por ello suele utilizarse como criterio de clasificación.

- *La Representatividad de una propiedad se utiliza para realizar inferencias.*

  La representatividad de las propiedades se utiliza en el proceso de equiparación para realizar inferencias, como se verá más adelante.

B.2) Criterios de Asignación de Representatividad a Propiedades

La asignación de la *Representatividad* a cada propiedad de un marco se realiza en base a:

* Otras propiedades detalladas en el mismo marco, y
* La representatividad de dicha propiedad en otros marcos.

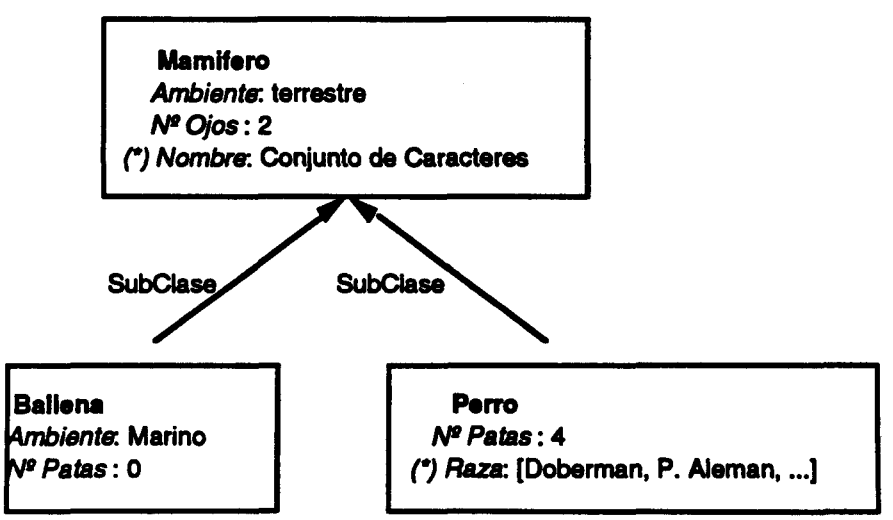


Figura 5.14. Análisis de Propiedades

Por ejemplo, supóngase una BC de *Mamíferos* y dos marcos clase de dicha BC llamados *Ballena* y *Perro* definidos según lo descrito en la figura 5.14. Al analizar la representatividad de las propiedades en ellos definidas se observa que:

- *La Representatividad es un valor local a cada marco*

  En una jerarquía pueden existir propiedades repetidas de clase o de instancia definidas en varios marcos, pero, en unos marcos son muy representativas y, en otros, son nada representativas. Por ejemplo, la propiedad de clase *Ambiente* en un marco *Mamífero* es menos representativa que la definición dada en el marco *Ballena*, pues existen muy pocos mamíferos que viven en un *Ambiente* marino y muchos que viven en la tierra.

- *Existen propiedades con el mismo nombre en diferentes marcos y, en cada uno de ellos, tienen una Representatividad diferente.*

  Por ejemplo, supóngase la propiedad de clase *Nº de Patas*. Dado que la mayoría de los mamíferos tienen cuatro patas, algunos dos patas y, solamente los cetáceos cero patas, se puede decir que, conocido el *Nº de Patas* de una entidad E, si ésta tiene cero patas es un cetáceo y si tiene dos o cuatro patas es imposible determinar la clase a la cual pertence la entidad. No obstante, la propiedad *Nº de Patas* definida en mamíferos que tengan dos patas es más representativa que la definición de dicha propiedad en mamíferos que tengan cuatro patas debido a que existen más clases de mamíferos con cuatro patas que con dos patas.

- *Existen propiedades con el mismo nombre en diferentes marcos y, en todos ellos, la propiedad es nada representativa.*

  Por ejemplo, la propiedad de clase *Numero de Ojos* es una propiedad que toma el valor *dos*. Conocer que un mamífero tiene dos ojos puede ser útil en una consulta, pero no aporta ninguna información en el proceso de equiparación, por tanto, la propiedad no es representativa.

  Un ejemplo en el que se muestra la ausencia de representatividad de una propiedad de instancia es el siguiente. Supóngase la propiedad *Nombre* definida en el marco *Mamífero* y dos entidades cuyos nombres son *Pluto* y *Moby Dick*. Un SBM, con esta información, sería incapaz de determinar la categoría en la

que se encuentra incluidas ambas entidades. Sin embargo, el conocimiento que permite a los humanos clasificar a *Pluto* como un perro y a *Moby Dick* como una ballena, es el conocimiento adicional sobre *Pluto* y sobre *Moby Dick*, conocimiento que, al no haberse introducido en la BC, origina importantes diferencias en las respuestas dadas por el ser humano y por el SBM.

Algo similar ocurriría si se supiera que el nombre de una entidad es *Felix*. En un país de habla fracesa se pensaría que *Felix* es un gato, mientras que en un país de habla castellana se pensaría que *Felix* es una persona. Todo ello lleva a afirmar que, si se toma esta propiedad como una propiedad que puede influir en la equiparación, se pueden cometer importantes errores al aplicar la herencia de propiedades.

- *Existen propiedades con el mismo nombre en diferentes marcos y, en todos ellos, la propiedad es muy representativa.*

  Por ejemplo, si los marcos *Perro* y *Ballena* de la figura 5.14 no se especializan en otros marcos y, en ambos marcos, se define la propiedad *Raza* como una propiedad de instancia que toma valores en un rango, entonces, conocida la raza de una entidad, se conoce si la entidad es instancia o no de la clase *Perro* o de la clase *Ballena*. Esto llevaría a definir la propiedad *Raza* en ambos marcos como una propiedad de instancia altamente representativa.

En definitiva, el estudio de estos casos lleva a establecer una clasificación de las propiedades de clase o de instancia que aparecen en cada marco clase en función de su representatividad en el concepto que el marco representa. Se necesita pues incorporar al Modelo de Diseño Basado en Marcos una faceta que contenga información sobre la representatividad de cada una de las propiedades definidas en cada marco de la BC. A esta faceta se la llamará *Representatividad*.

La *Representatividad* de cada propiedad permitirá al IC introducir en cada Modelo Formalizado Basado en Marcos conocimiento que antes no introducía al construir la BC y que sin embargo sí poseía el experto. De esta forma, se incrementa la expresividad del formalismo y se disminuye la pérdida de conocimiento que se produce al formalizar el conocimiento del *Mundo Externo*. De nuevo, con esta mejora, se ha acercado el *Modelo Formalizado* Basado en Marcos al *Mundo Externo*, al incrementar la expresividad tratada por el Modelo de Diseño.

B.3) Definición de la Representatividad

Se define la propiedad P como una **propiedad más representativa** que la propiedad Q en un marco clase si, conocidos los valores de P y de Q en una entidad E, la propiedad P proporciona mayor información que la propiedad Q en la descripción del concepto que el marco clase representa. Por ejemplo, la propiedad *Número de Patas* definida en el marco *Perro* es más representativa que la propiedad *Nombre*, pero menos que la propiedad *Ladra*.

B.4) Cuantificación de la Representatividad

El aporte de significado que cada propiedad proporciona en un marco clase lo proporciona el experto al IC al construir la BC. En el Modelo de Diseño Orientado a Marcos que se propone, esta medida toma valores con dos decimales en el intervalo [0,1] con el siguiente significado:

* Si la *Representatividad* de una propiedad toma el valor cero, entonces, la propiedad es meramente descriptiva en el marco en el que se ha definido y no discrimina entidades de la clase de aquellas que no lo son.

* Si la *Representatividad* de una propiedad toma cualquier valor comprendido entre cero y uno, entonces, la propiedad discrimina en el valor asignado a las instancias de la clase.

* Si es uno, la propiedad, en el Modelo de Diseño, recibe el nombre de **Propiedad Específica**. Conocida la presencia de una propiedad P en una entidad E, si esta propiedad es específica de un marco clase, entonces, la entidad se puede clasificar como una instancia de dicha clase. Ejemplos de propiedades específicas de los marcos *Perro* y *Gato* son, respectivamente, las propiedades *Ladra* y *Maulla*.

  En cualquier marco clase se puede definir un número cualquiera de propiedades específicas. No obstante, la clasificación de la entidad en la clase se produce nada más conocer, con total seguridad, la presencia de una de estas propiedades en la entidad. Por este motivo, el IC debe actuar con cautela a la hora de asignar representatividades con valor 1 a las propiedades de una clase, comprobando con el experto que la presencia de la propiedad en la entidad es condición suficiente, aunque no necesaria,

para clasificarla en la clase. Por ejemplo, si en el marco clase *Perro* se definen las propiedades *Ladra* y *Raza* como propiedades específicas (representatividad = 1) de la clase, dada una entidad E para la cual se conoce la raza, si ésta está incluida en los valores definidos en la propiedad *Raza* del marco clase *Perro,* se puede decir, aunque no se conozca si la entidad ladra o no, que se trata de un perro. Igual sucedería si se conociera con total seguridad que la entidad ladra y se desconociera su raza.

* Si el IC no introduce la *Representatividad* de algunas o de todas las propiedades, por omisión, la *Representatividad* de las propiedades no definidas toma el valor 0,2. Empíricamente se ha demostrado que, suponiendo las representatividades de todas las propiedades de la BC desconocidas, el resultado de la equiparación de una entidad E con un marco clase MC de la BC es parecida a la que realizaría un humano en la misma situación si se asigna este valor a la representatividad de cada propiedad desconocida.

Por ejemplo, supóngase como representatividades de las propiedades de la jerarquía de la figura 5.14 las mostradas en la figura 5.15.

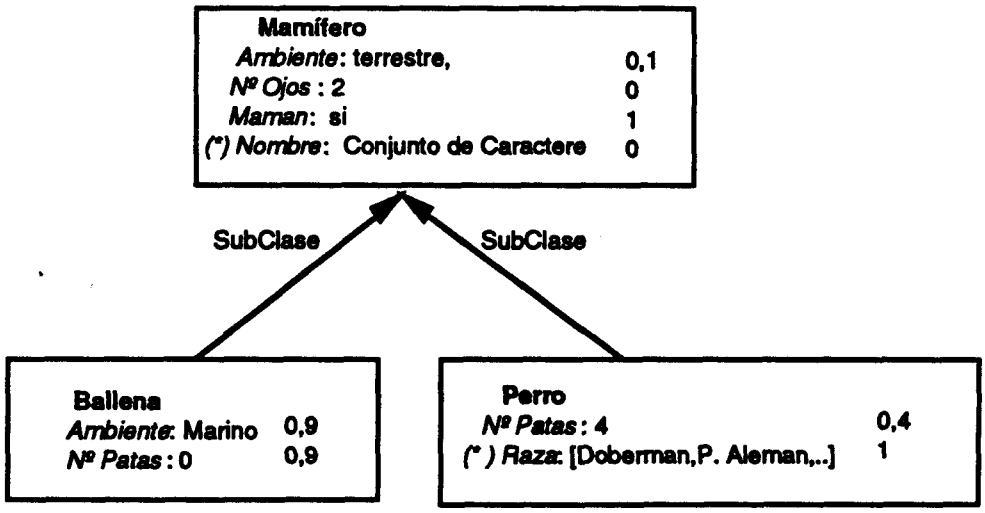


Figura 5.15. Asignación de Representatividad.

Como conclusiones a este apartado hay que decir que:

* La *Representatividad* de una propiedad es un criterio subjetivo proporcionado por el experto.

* La *Representatividad* es una medida nueva que va a utilizar el IC al construir BB.CC. basadas en marcos para definir el aporte discriminitario que tiene una propiedad dentro de un marco.

* La *Representatividad* incrementa la expresividad del Modelo Formalizado al introducir en él conocimiento de sentido común presente en el *Mundo Externo* que antes no se podía modelizar en las BB.CC. basadas en marcos.

* El IC debe comprobar antes de asignar un valor de representatividad máximo a una propiedad de un marco clase que la presencia de dicha propiedad en una entidad es condición suficiente, aunque no necesaria, para clasificar la entidad en la clase. En caso de duda se debe asignar un valor alto, por ejemplo 0,8, distinto de 1.

* El IC puede definir cualquier número de propiedades específicas en un marco clase. La presencia de una de ellas es determinante en la clasificación de la entidad en la clase.

5.2.2.2.2. Características de las Propiedades: Facetas

Identificadas las propiedades de los marcos clase, es necesario definir, para cada propiedad, sus atributos. Dado que son múltiples los usos que se pueden hacer de una propiedad, también son diversas las formas en las que se pueden definir cada una de estas propiedades. Las **facetas**, o propiedades que definen propiedades, van a permitir especificar sus peculiaridades. En este trabajo se proponen dos clasificaciones combinadas de las facetas. La primera, se basa en el tipo de conocimiento que almacenan y, la segunda, en el tipo de propiedad en el que se definen. El esquema, que a continuación se propone a la figura 5.16, ha sido el utilizado en la exposición de este apartado:

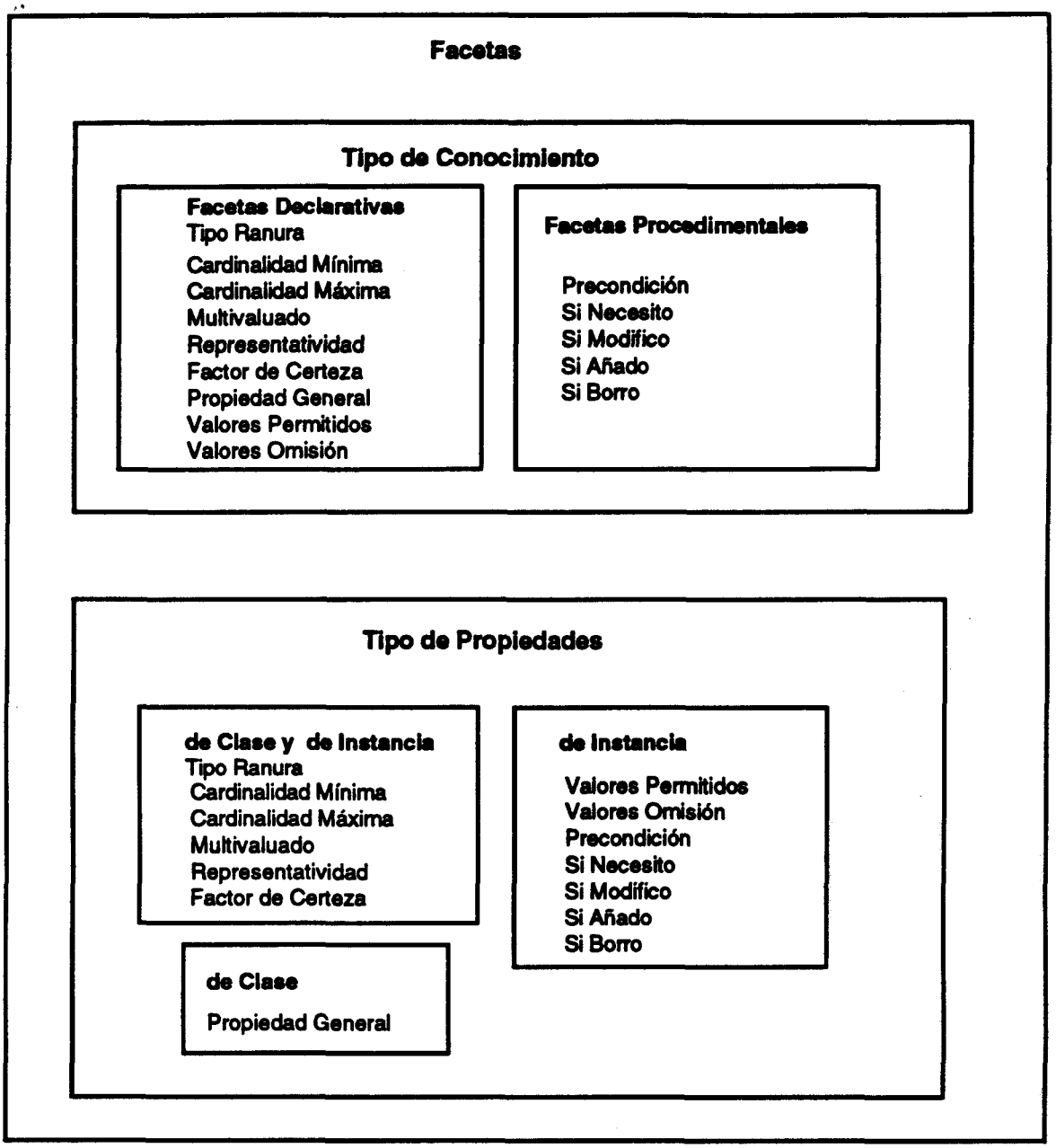


Figura 5.16. Tipos de Facetas.

Dado que cualquier propiedad de Clase o de Instancia puede haber rellenado sus facetas con definiciones de valores declarativos o procedimentales, se ha decidido analizar primero las facetas desde el punto de vista del tipo de conocimiento con el que se rellenan y, posteriormente, en función del tipo de propiedad en la que se definen.

En esta sección, se proponen un conjunto de "recetas" que debe utilizar el IC a la hora de definir las facetas de una propiedad. Es decir, para cada tipo de propiedad, se propondrán sus facetas específicas y las definiciones declarativas y procedimentales

permitidas en cada una de ellas. Con ello, se consigue una estandarización en la definición de las propiedades, y se facilita al IC la labor de definición de dichas propiedades en el Modelo Formalizado. El Modelo de Diseño incorpora esta estandarización con el fin de impedir la definición incorrecta de los valores de las propiedades en los Modelos Formalizados.

A) FACETAS DECLARATIVAS Y PROCEDIMENTALES.

Declarativamente, una propiedad se define en el Modelo de Diseño especificando el tipo básico de valores con los que se puede rellenar. Ejemplos de definiciones declarativas de propiedades serían definir una propiedad de tipo: lógico, carácter, número, entero, real, o como un conjunto restringido de valores numéricos, caracteres, etc.

Procedimentalmente, una propiedad se define especificando un procedimiento o una regla. En el Modelo de Diseño que se propone en este trabajo el conocimiento procedimental se expresa únicamente utilizando procedimientos. Con el fin de garantizar su correcta ejecución, en cada procedimiento se definirán unas Precondiciones y unas Postcondiciones. La expresión:

Precondición Acción Postcondición

importada de la Ingeniería del Software, tiene el siguiente significado:

> *Si la ejecución de la Acción comienza en un estado que satisface las Precondiciones, entonces, garantiza que la Acción termina siempre en un tiempo finito en un estado que satisface las Postcondiciones.*

Introducir este tipo de conocimiento en la definición de un procedimiento es importante porque obliga al IC a definir un conjunto de condiciones que garantizan la perfecta ejecución del procedimiento. Si las condiciones no se cumplen, el procedimiento no se ejecuta. Este aspecto es especialmente importante sobre todo si se tiene en cuenta que las inferencias asociadas al conocimiento procedimental son muy poderosas en los marcos y que apenas se utilizan por la poca garantía que ofrecen.

B) TIPO DE PROPIEDAD.

Las facetas, independientemente de que se rellenen con valores declarativos o procedimentales, en este Modelo se clasifican en tres categorías:

a) Las que se utilizan en la definición de cualquier propiedad, con independencia de si la propiedad es de clase o de instancia,

b) Las que solamente se utilizan para definir propiedades de clase, y,

c) Las que solamente se utilizan para definir propiedades de instacia.

a) Facetas de propiedades de clase y de instancia

En cualquier propiedad, independientemente de que se trate de una propiedad de clase o de instancia habrá que definir obligatoriamente las facetas: Tipo Ranura, Cardinalidad Mínima, Cardinalidad Máxima y Multivaluada y, opcionalmente, las de Representatividad y Factor de Certeza.

Tipo Ranura

Es una faceta declarativa que se utiliza para comprobar que, una vez rellena la ranura con un valor concreto, el valor pertenece al tipo especificado en ella. Esta faceta se define con valores declarativos o con punteros a otros marcos:

a) Si se rellena con valores declarativos, deberá especificarse si los valores son enteros, reales, etc.

b) Si se rellena con punteros a otros marcos, se indicará el nombre del marco clase con el que se rellena.

Cardinalidad Mínima

Es una faceta delarativa que especifica el número mínimo de valores con los que se puede rellenar la propiedad que se está definiendo en un marco cualquiera del SBM. Se tiene, por tanto, una restricción en el número de valores mínimos con los que debe rellenarse la propiedad.

Cardinalidad Máxima

La Cardinalidad Máxima es una faceta declarativa que representa el número máximo de valores con los que se puede rellenar la propiedad en un marco. Al igual que sucedía con la cardinalidad mínima, esta faceta proporciona una restricción en el número de valores máximos con los que simultáneamente puede rellenarse la propiedad.

Multivaluada

Faceta declarativa que determina si la ranura se puede rellenar con más de un valor o no. Este dato se puede obtener utilizando un procedimiento que, en función del valor almacenado en la ranura Cardinalidad Máxima, introduzca en esta ranura el valor sí o no. Por tanto, el valor con el que se rellena la faceta es un valor declarativo, sin embargo, la definición de la faceta puede ser procedimental.

Representatividad

Faceta declarativa en la que el IC introduce el valor de la *Representatividad* de la propiedad en el marco clase en cuestión.

Grado de Certeza.

Es una faceta declarativa que almacenará el grado de certeza que tiene el usuario de la aplicación sobre el valor de la propiedad P en una entidad E.

b) Facetas de Propiedades de Clase

Si lo que se está definiendo es una propiedad de clase en un marco clase, habrá que especificar, para cada propiedad, una faceta llamada **Propiedad General** que se rellenará con los valores declarativos o procedimentales que asignan valores a la propiedad en el marco.

c) Facetas de Propiedades de Instancia

Para cada una de las propiedades de instacia que se definen en un marco clase, habrá que rellenar las facetas que a continuación se citan:

Valores Permitidos

Faceta declarativa en la que se especifican los valores permitidos con los que se puede rellenar la ranura. Los valores pueden definirse utilizando conocimiento declarativo o procedimental. Concretamente, se utilizará:

a) Un *Rango de Valores*,

b) Un *Tipo de Datos Básicos*,

c) Un *Procedimiento*

d) Un *Puntero a un Marco*

Independientemente de cómo se defina la propiedad, todas las definiciones deben conducir al mismo tipo básico, es decir, a un tipo entero, a un tipo carácter, a un marco clase, etc.

Valor por Omisión

El (los) valor(es) por omisión son los valores con los que se rellena una ranura de un marco instanciado si no se conoce de forma explícita otro valor. Si la propiedad que se está definiendo es simple se define un único valor, si es multivaluada, se pueden definir varios. En cualquier caso, el (los) valor(es) asignado(s) será(n) del tipo especificado en la faceta Tipo Ranura.

Precondición

Faceta procedimental que define los prerrequisitos que deben cumplir los valores asociados a esta propiedad en el Marco Clase en el que se define. Las condiciones se pueden expresar de la siguiente forma:

a) Utilizando expresiones matemáticas que acoten el valor que toma la ranura,

b) Utilizando un procedimiento que comprueba que el valor de la ranura es correcto, y,

c) Refiriéndose a valores que toman otras ranuras en otros marcos.

SI Necesito

Faceta procedimental que almacena los procedimientos o reglas que deben ejecutarse cuando se *necesite* obtener el valor de una propiedad en un marco instanciado. Este procedimiento puede preguntar el dato al usuario del sistema o bien puede calcular el dato a partir de otros datos implícitamente almacenados en la BC.

Al ser una faceta procedimental se define utilizando el esquema de precondición postcondición anteriormente expuesto.

SI Añado

Faceta procedimental que almacena los procedimientos o reglas que deben ejecutarse cuando se *añada* un dato a la ranura que representa la propiedad en cuestión en un marco instanciado.

SI Modifico

Faceta procedimental que almacena los procedimientos o reglas que deben ejecutarse cuando se *modifica* un dato en una ranura de un marco instanciado.

SI Borro

Faceta procedimental que almacena los procedimientos o reglas que se ejecutan cuando se *borra* un dato en una ranura de un marco instanciado.

5.2.3 LA REPRESENTACION TABULAR DE UN MARCO

La tabla que a continuación se muestra sintetiza todos los conceptos expuestos en el apartado 5.2.2.2.2. Supóngase que M es un marco cualquiera de un SBM que está conectado mediante relaciones estándares y no estándares con otros marcos del SBM. En M se han definido un conjunto de propiedades de clase, otras de instancia y un conjunto de métodos. Si cada fila de la tabla representa los tipos de ranuras anteriormente citadas y las columnas, sus facetas, la tabla 5.1 representa el tipo de conocimiento con el que se rellena cada faceta asociada a cada tipo de ranura.

TIPO RANURA	CARD. MINIMA	CARD. MAXIMA	MULTIVAL.	REPRESENTATIVIDAD	GRADO DE CERTEZA	PROPIEDAD GENERAL	VALORES PERMITIDOS	VALORES OMISION	PRECONDICION	SI NECESITO SI AÑADO SI MODIFICO SI BORRO
Relaciones	Natural	Natural	[Cierto, Falso]	------	------	------	------	------	------	------
Propiedades de Clase	Natural	Natural	[Cierto, Falso]	[0,1]	[-1, 1]	Valor Simple Valor Compuesto	------	------	------	------
Propiedades de Instancia	Natural	Natural	[Cierto, Falso]	[0,1]	[-1, 1]	------	Rango de Valores Tipo de Datos Procedimiento Puntero	Valor Simple Valor Compuesto	Ecuación Procedimiento	Procedimiento Regla
Métodos	1	1	Falso	------	------	Procedimiento Regla	------	------	------	------

Tabla 5.1. Síntesis de todos los conceptos

5.3. RAZONAMIENTO: TEORIA DE MARCOS.

En el Modelo de Diseño Orientado a Marcos que aquí se presenta, se han incluido ciertos elementos en el tratamiento de las técnicas clásicas de inferencia en Marcos y se han introducido otras técnicas nuevas. Las novedades más significativas son:

1. La propuesta de **Equiparación** trabaja con el conocimiento incierto del dominio y con las representatividades dadas por el experto e introducidas por el IC en cada propiedad de cada marco clase con el que una entidad dada intenta equipararse. El valor de equiparación informa del grado de compatibilidad entre la clase y la entidad.

2. La transferencia de propiedades realizadas sobre relaciones "ad hoc" definidas por el IC al construir el sistema, es una técnica de inferencia independiente y complementaria a la cesión de propiedades llamada **Herencia**. En este trabajo se propone llamar a esta nueva técnica que transfiere propiedades entre marcos conectados por relaciones "ad hoc", **Donación**. La propuesta de cesión de propiedades combina ambas técnicas al realizar las inferencias.

5.3.1. EQUIPARACION

Conocidos los valores de un conjunto de propiedades que describen parcialmente una entidad E y sus certezas asociadas, la técnica de inferencia basada en equiparación tiene que encontrar el marco clase de la BC más consistente en su descripción con la descripción de la situación actual dada por la entidad. Encontrado el/lo(s) marco(s) clase, la entidad se convierte en un marco instanciado de dicho(s) marco(s) clase. El problema es el siguiente, si los valores de todas las propiedades de una entidad E fueran conocidos y dichas propiedades se encontraran almacenadas en un único marco clase de la jerarquía, la equiparación de la entidad con el marco clase sería perfecta. Sin embargo, esta situación raramente ocurre debido a que:

1. El conocimiento se encuentra distribuído en el SBM por todos los marcos de la jerarquía.

2. No todas las propiedades de una entidad E son conocidas al empezar la equiparación. El conocimiento que se posee de la entidad que se está

clasificando al realizar la equiparación es limitado. Es decir, se conoce un conjunto finito, no muy amplio, de sus propiedades.

3. Aun siendo conocidos los valores de algunas propiedades en una entidad E, se pueden tener distintos grados de seguridad sobre los valores que toman dichas propiedades en la entidad E. Por ejemplo, para una entidad se puede conocer con certeza 0,5 (seguridad media sobre el valor dado) que dicha entidad *tiene pulmones*, y con certeza 1 (seguridad completa sobre el valor) que su *número de patas es cuatro* o que *no es un Pekinés*.

4. No todas las propiedades aportan el mismo grado de conocimiento a la hora de realizar la equiparación. Por ejemplo, en una BC sobre animales, si se conoce que una entidad E *Mama* y vive en *Ambiente Terrestre*, un motor de inferencia que utilizara equiparación sería incapaz de determinar en qué categoría se incluiría dicha entidad. Sin embargo, si la entidad E *Mama* y vive en *Ambiente Marino*, el motor de inferencia realizaría una equiparación perfecta si dijera que la entidad E se equipara con el marco clase *Cetáceo*, convirtiendo a la entidad en una instancia de dicha clase. En este caso, se estaría utilizando la representatividad de la propiedad *Ambiente* de los marcos *Mamífero* y *Cetáceo* para determinar a qué clase pertenece la entidad.

En este trabajo, se propone una técnica de equiparación que trabaja con el conocimiento incierto presente en el dominio. Esta técnica de equiparación, utiliza la representatividad de las propiedades de los marcos clase, la certeza de los valores de las propiedades de las entidades y la herencia de propiedades para calcular el valor de equiparación de una entidad E con un marco clase. Este valor determina en qué medida los marcos clase seleccionados son consistentes con la descripción de la situación actual dada por la entidad.

Básicamente, los pasos en los que se descompone el proceso de equiparación de una entidad E con un marco clase se muestran gráficamente en la figura 5.17 y son:

1. Asignación del grado de certeza a los valores con los que se rellenan las propiedades definidas en la entidad E.

2. Selección del conjunto de marcos clase candidatos con los que la entidad E se puede equiparar.

3. Para cada marco clase candidato, se calcula el Valor de Equiparación (VE) de la entidad en él.

4. Selección del o de los marco(s) clase equiparable(s).

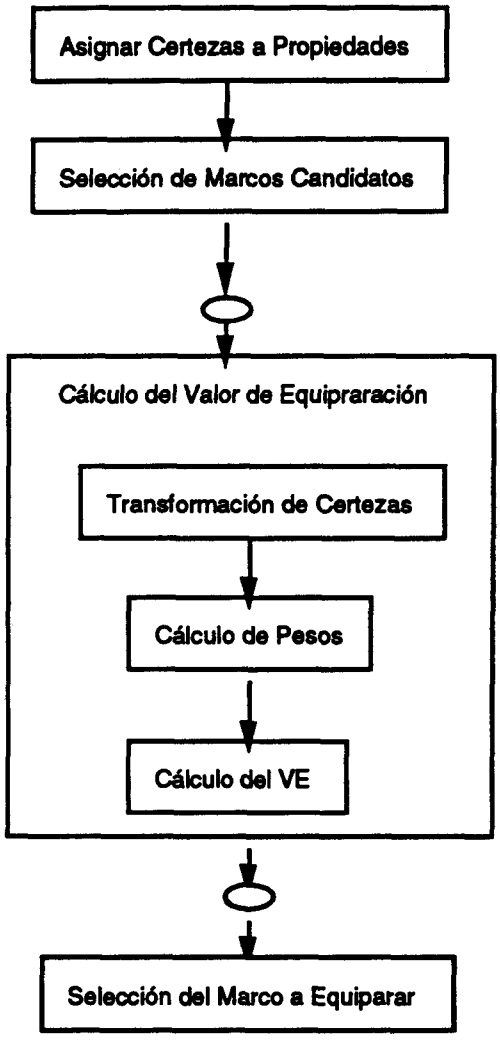


Figura 5.17. Pasos en el proceso de Equiparación

En la descripción de cada uno de estos pasos se ha tomado como BC la mostrada en la figura 5.18.

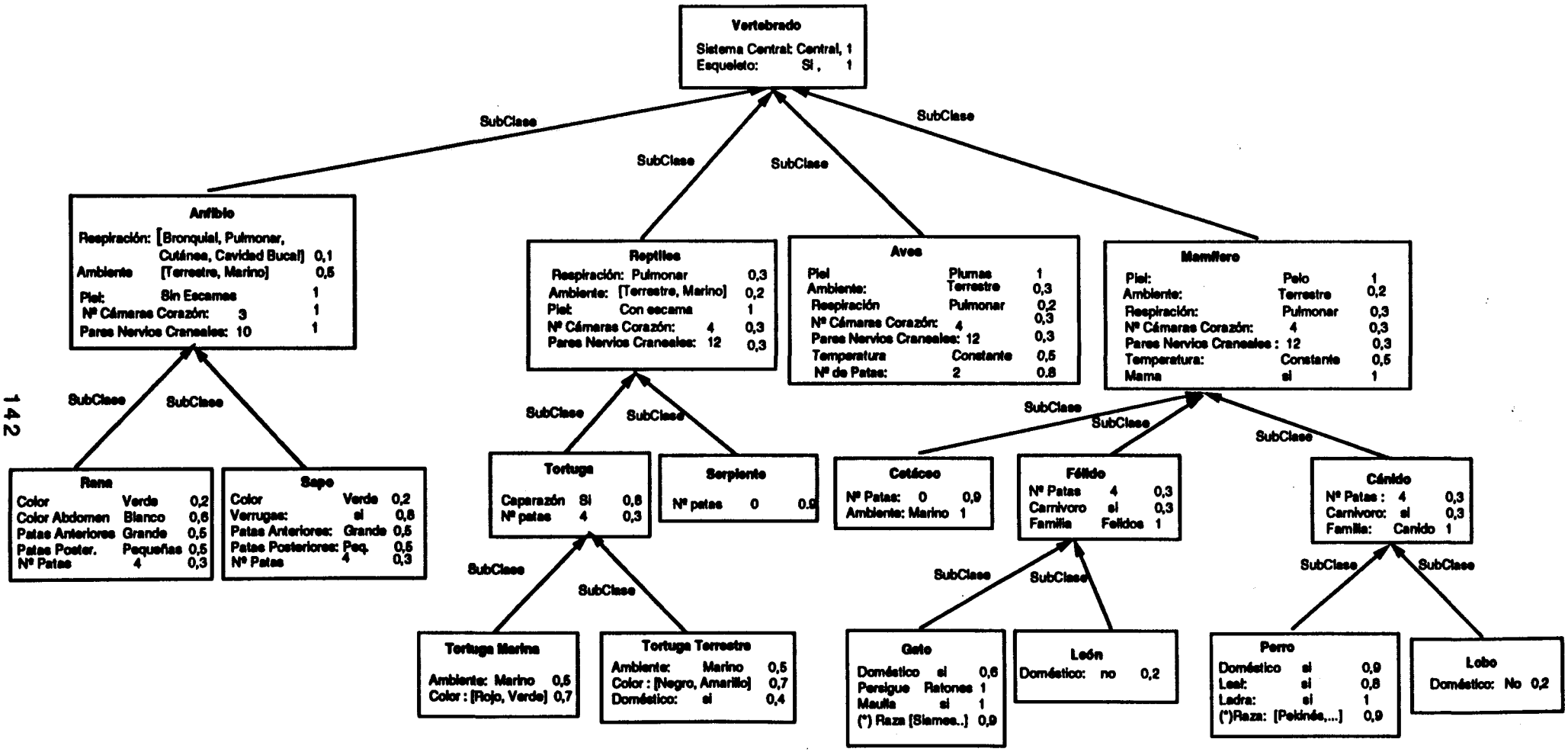


Figura 5.18. Jerarquía de Vertebrados

5.3.1.1. Asignación de Certeza a Propiedades

En el tratamiento del grado de certeza que presentan los valores de una propiedad P en una entidad se han hecho las siguientes suposiciones:

1. El grado de certeza *c* de una propiedad P en una entidad la proporciona, opcionalmente, el usuario del SBC nada más introducir los valores de la propiedad P en la entidad.

2. La certeza *c* es un número comprendido en el intervalo [-1,1] que indica la seguridad con la que el usuario conoce el valor de una propiedad P en una entidad E. En la figura 5.19, se muestran los valores asignados a la certeza.

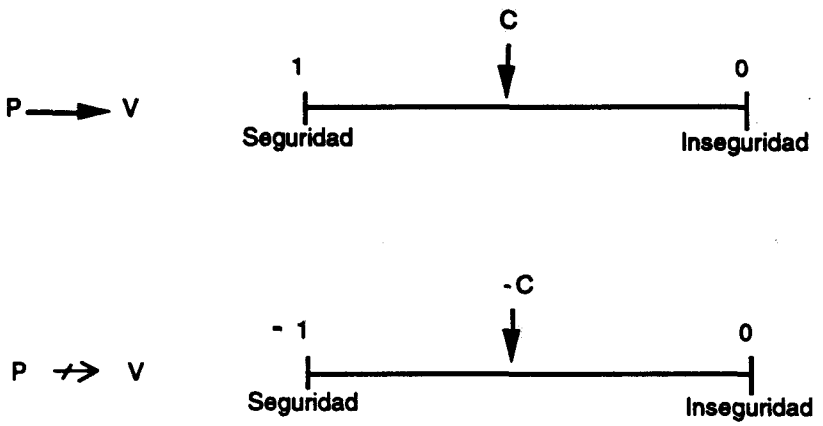


Figura 5.19. Significado de la Certeza.

En este trabajo el significado de esta medida es el siguiente:

2.1. Si la certeza toma el valor uno, el usuario tiene una seguridad absoluta en el valor V con el que se ha rellenado la propiedad P en la entidad E.

2.2 Si la certeza toma un valor comprendido en el intervalo abierto (0,1), entonces, la certeza del valor V con el que se ha rellenado la propiedad P en la entidad es función de dicho valor.

2.3 Si la certeza toma el valor cero, entonces, el usuario no tiene ninguna seguridad, ni a favor ni en contra, en el valor V asignado a la propiedad P de la entidad E.

2.4 Si la certeza toma valores en el intervalo abierto (-1,0), entonces, se conoce con el valor de certeza asignado que la entidad E *no* posee el valor V en la propiedad P.

2.5 Si la certeza toma el valor -1, entonces, el usuario está totalmente seguro de que la entidad no posee el valor V en la propiedad P, pero no se conocen los valores que realmente toma dicha propiedad.

3. La precisión máxima permitida para la certeza *c* es de dos decimales.

4. El valor por omisión asociado a la certeza de los valores de cualquier propiedad P almacenada en una entidad E es 0,2. Empíricamente se ha demostrado que asignado este valor por omisión a la certeza, los resultados obtenidos al final del proceso de equiparación se aproximan a las respuestas que daría un ser humano en las mismas circustancias.

Por ejemplo, supóngase que el IC ha observado una entidad con las siguientes propiedades y grados de certeza:

E1	Valor	Certeza
Esqueleto	Si	1
Mama	Si	1
Respiración	Pulmonar	0,8
Nº Patas	4	1
Ambiente	Marino	-1
Raza	Pekinés	-0,5

El significado de los valores de certeza de las propiedades de la entidad E1 es el siguiente:

* El usuario está totalmente seguro de que la propiedad *Esqueleto* toma el valor *sí*, que la propiedad *Mama* también, y que el *Nº de Patas* de la entidad es *cuatro.*

* El usuario tiene seguridad alta sobre que la entidad tiene *Respiración Pulmonar.*

* El usuario está parcialmente seguro de que la propiedad *Raza* no toma el valor *Pekinés*, pero desconoce los valores que puede tomar la propiedad.

* El usuario está totalmente seguro de que la entidad no vive en *Ambiente Marino*.

5.3.1.2. Selección de Marcos Candidatos

La selección del conjunto de marcos clase candidatos con los que la entidad E se puede equiparar, se realiza en este trabajo buscando en la BC marcos clases en los que al menos se encuentre definida una propiedad conocida en la entidad. Aquellos marcos clase que no tengan como mínimo una propiedad común con la entidad, no son equiparables.

Por ejemplo, dada la BC de la figura 5.18, el conjunto de marcos clase equiparables con la entidad E1 serían todos: *Vertebrado*, *Anfibio*, *Reptiles*, *Aves*, *Mamíferos*, *Ranas*, *Sapos*, *Tortugas*, *Tortugas Terrestres*, *Tortugas Marinas*, *Serpiente*, *Cetáceo*, *Félidos*, *Cánidos*, *Gato* y *Perro*.

5.3.1.3. Cálculo del Valor de Equiparación

Realizada la selección del conjunto de marcos candidatos con los que la entidad E se puede equiparar, se pasa a calcular el valor de equiparación de la entidad E en cada uno de ellos.

5.3.1.3.1. Definición del Valor de Equiparación

El VE es una métrica cuantitativa que determina en qué medida se puede afirmar que un marco clase es consistente con la descripción de la situación actual dada por la entidad E. El VE es un número comprendido en el intervalo [-1, 1]. El significado de esta medida en este trabajo es el siguiente:

1. Si el VE es cero o es negativo, se tiene una inconsistencia en la equiparación de la entidad E con el marco clase en cuestión. Bajo ningún concepto la entidad E se convierte en una instancia de dicha clase.

2. Si es uno, significa que la equiparación es perfecta, es decir, totalmente consistente. En este caso, el marco clase es uno de los marcos seleccionados en el proceso de equiparación y la entidad E puede convertirse en un marco instanciado de dicha clase.

3. Si está comprendido entre cero y uno, el grado de consistencia en la equiparación es directamente proporcional al valor de equiparación obtenido y la entidad E puede convertirse en un marco instanciado del marco clase con VE más alto.

5.3.1.3.2. Métrica del Valor de Equiparación

Para cada uno de los marcos candidatos seleccionados en el paso anterior, se calcula el VE de la entidad E en ellos. Supóngase un marco clase MC en el que se han definido un conjunto de propiedades con sus valores y representatividades asociados, como gráficamente se muestra en la figura 5.20.

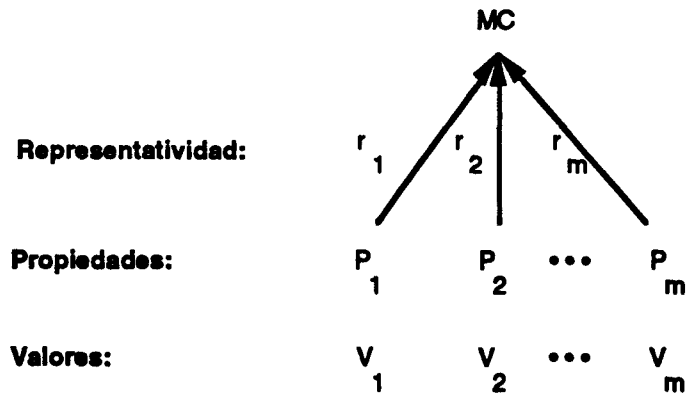


Figura 5.20. Propiedades, valores y representatividades en un marco clase.

Para cada marco clase se realizan las operaciones que a continuación se citan en el cálculo del VE.

A) Transformar las certezas asociadas a las propiedades de una entidad E en certezas de las propiedades de la clase a equiparar.

B) Para cada propiedad definida en el conjunto de marcos clase seleccionados, se calcula su *Peso*. Si la propiedad no se encuentra

definida en el marco, aplicando herencia, se busca la propiedad en sus ascendientes. En caso de encontrarla, se adapta el peso al marco con el cual se intenta la equiparación.

C) Para cada marco clase seleccionado se calcula el VE en función de los pesos anteriores.

A) Transformación de las Certezas

En este apartado se transforman las certezas asociadas a las propiedades de la entidad en certezas asociadas a las propiedades de la clase con la que se intenta equiparar. Se pueden dar tres casos que, abreviadamente, se representan en la figura 5.21, y son:

1. La propiedad P toma el valor V con certeza positiva: $P \rightarrow V$
2. La propiedad P toma el valor V con certeza negativa: $P \not\rightarrow V$
3. La propiedad P es desconocida en la entidad.

1. La propiedad P toma el valor V con certeza positiva: $P \rightarrow V$

Si la certeza asociada al valor V de la propiedad P de la entidad E toma valores comprendidos en el intervalo [0, 1], entonces, la asignación de certezas a la clase se realiza de acuerdo a las siguientes reglas:

1.1 Si el valor con el que se rellena la propiedad P en la entidad está incluido o es igual a el/los valor(es) permitido(s) o asignado(s) a la propiedad P en el marco clase, entonces, la certeza de los valores de la propiedad en el marco clase es la proporcionada por el usuario en la entidad o el valor asignado por omisión.

Por ejemplo, al transformar la certeza de la propiedad *Nº de Patas* de la entidad E1 en certeza de la propiedad *Nº de Patas* en la clase *Cánidos*, *Félidos*, *Rana*, *Sapo* y *Tortuga*, el valor que toma la certeza de esta propiedad en cada clase es el proporcionado por el usuario. En este caso, en cada marco clase se rellenaría la certeza asociada a la propiedad *Nº de Patas* con el valor uno.

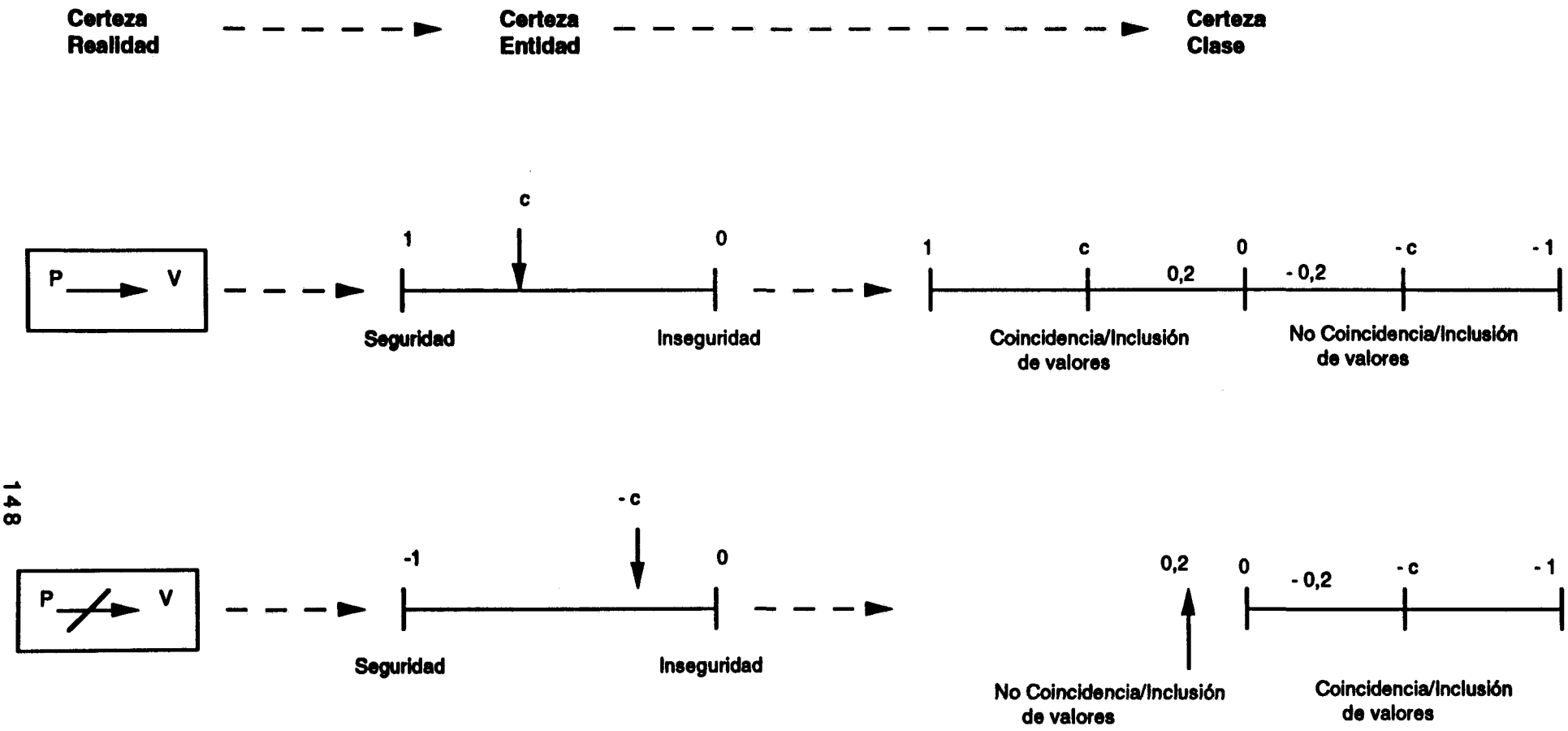


Figura 5.21. Transformación de Certezas

1.2 Si el valor asignado a la propiedad P en la entidad no está incluido o no es igual a el/los valor(es) permitido(s) o asignado(s) a la propiedad P en el marco clase, entonces, se tiene certeza en contra en la equiparación de la entidad como una instancia del marco clase. Por este motivo, la certeza de la propiedad P en la entidad, se convierte en una certeza negativa en la clase, tomando el valor *-c* o la negación del valor asignado por omisión.

Así, al intentar equiparar la entidad E1 con los marcos clase *Cetáceos* y *Serpiente*, la certeza que toman la propiedad *Nº de Patas* en dichos marcos es el valor negado de la certeza asignada por el usuario en la propiedad de la entidad. En este caso, para los marcos clase, anteriormente mencionados, la certeza de la propiedad *Nº de Patas* en cada marco clase se rellenaría con el valor menos uno.

2. *La propiedad P toma el valor V con certeza negativa: P $\not\rightarrow$ V*

Si la certeza asociada al valor V en la propiedad P de la entidad E toma valores comprendidos en el intervalo [-1, 0], entonces, se sabe, con una determinada certeza, que la propiedad P no toma el valor V en la entidad E, pero se desconocen los valores que realmente puede tomar esta propiedad y que tendrían asignados una certeza positiva. La transformación de la certeza negativa en certezas positivas se realizan de acuerdo a las siguientes reglas.

2.1 Si los valores de la propiedad en la entidad coinciden, o están incluidos en los valores de la propiedad de la clase, entonces, se mantiene la certeza negativa, pues se sabe explícitamente, por el valor negativo de la certeza, que la propiedad no aporta información a favor de la equiparación de la entidad con la clase.

Por ejemplo, si se sabe con seguridad media que la *Raza de* una entidad E *no* es *Pekinés*, entonces, al intentar la equiparación de la entidad E con el marco *Perro*, el valor de certeza negativo se mantiene pues se conoce parcialmente que esta entidad no es un *Perro*.

2.2 Si los valores de la propiedad de la entidad no coinciden, o no están incluidos en los valores de la propiedad de la clase, entonces, no se

conoce en qué medida la entidad pertenece o no a la clase. Por ello, se asigna el valor por omisión a la certeza de la propiedad en la clase al desconocerse los valores que toma la propiedad y la certeza asignada a dichos valores.

Sin embargo, al intentar equiparar la entidad anterior con el marco clase: *Gato*, *Lobo*, *León*, o *Tortuga Terrestre*, el sistema no es capaz de discriminar si la entidad pertenece o no a estas clases, pues el único dato que tiene el sistema es que la entidad no es un *Perro*. Por este motivo, se asigna a la certeza de la propiedad *Raza* de los marcos anteriormente mencionados el valor por omisión.

3. *La Propiedad P es desconocida en la entidad*

El resto de propiedades desconocidas en la entidad, pero definidas en el marco clase, tendrán certeza cero en la clase, pues no influyen ni a favor ni en contra en la equiparación de la entidad con la clase.

Supóngase la entidad E1 anterior y la BC de *Vertebrados* de la figura 5.18. En la tabla 5.2 se muestra la transformación de la certeza de cada propiedad en certeza de la propiedad en cada una de las clases con las que la entidad E1 se puede equiparar según los criterios establecidos en esta sección.

B) El Peso de una Propiedad: Certeza y Representatividad

Convertida la certeza de una propiedad P de una entidad E en una certeza a favor o en contra en el marco clase con el que se intenta equiparar, y conocida la representatividad de la propiedad P en dicho marco, es el momento de combinar la certeza con la representatividad para calcular el *Peso* que tendría la propiedad P en la equiparación de la entidad con el marco clase.

Básicamente, en la técnica que se propone, se pueden dar dos situaciones en el cálculo del peso de las propiedades conocidas en la entidad E en el marco Clase:

1. Si la propiedad P, se encuentra definida en el marco clase con el que se intenta la equiparación, entonces, se calcula el peso de la propiedad P en dicho marco como:

$$Peso_p = Certeza_p * Representatividad_p \qquad (1)$$

Entidad		
Propiedad	**Valor**	**Certeza**
Esqueleto:	Si	1
Mama:	Si	1
Respiración:	Pulmonar	0,8
Nº de Patas:	4	1
Ambiente:	Marino	-1
Raza:	Pekinés	-0,5

Marco Clase	Propiedad	Valor	Certeza	Criterio
Vertebrado	*Esqueleto:*	Si	1	P →V
Anfibio	*Respiración:*	Pulmonar, Bronquial, Cutánea, Pulmonar	0,8	P →V
	Ambiente:	Marino, Terrestre	-1	P ↛V
Reptil	*Respiración:*	Pulmonar	0,8	P →V
	Ambiente:	Marino, Terrestre	-1	P ↛V
Ave	*Respiración:*	Pulmonar	0,8	P →V
	Ambiente:	Terrestre	0,2	P ↛V
	Nº de Patas:	2	-1	P →V
Mamífero	*Mama:*	Si	1	P →V
	Respiración:	Pulmonar	0,8	P →V
	Ambiente:	Terrestre	0,2	P ↛V
Rana	*Nº de Patas:*	4	1	P →V
Sapo	*Nº de Patas:*	4	1	P →V
Tortuga	*Nº de Patas:*	4	1	P →V
Tortuga Terrestre	*Ambiente:*	Terrestre	0,2	P ↛V
Tortuga Marina	*Ambiente:*	Marino	-1	P ↛V
Serpiente	*Nº de Patas:*	0	-1	P →V
Cetáceo	*Nº de Patas:*	0	-1	P →V
	Ambiente:	Marino	-1	P ↛V
Félido	*Nº de Patas:*	4	1	P →V
Cánido	*Nº de Patas:*	4	1	P →V
Gato	*Raza:*	[Siamés, ...]	0,2	P ↛V
Perro	*Raza:*	[Pekinés, ...]	-0,5	P ↛V

Tabla 5.2. Transformación de las Certezas

2. Si la propiedad P, no se encuentra definida en el marco clase MC_1 con el que se intenta la equiparación, entonces, si dicha propiedad se encuentra en el marco clase MC_2, marco accesible desde el marco clase MC_1 aplicando herencia, se calcula el peso de la propiedad P en el marco MC_1 de acuerdo a las siguientes reglas:

2.1) Si la certeza de la propiedad P en la nueva clase MC_2 es positiva, el peso de la propiedad en el marco MC_1 se calcula como:

$$Peso_P = \frac{(Certeza_P * Representatividad_P)}{número\ de\ arcos + 1} \qquad (2)$$

Se introduce el factor corrector (Nº de arcos +1) porque propiedades muy representativas en marcos clase van perdiendo representatividad a medida que la clase se especializa en otros marcos clase. Es decir, la representatividad de una propiedad en un marco clase, se va perdiendo o difuminando en la jerarquía a medida que dicha clase se va especializando en otras clases. No obstante, aunque el peso obtenido sea mínimo, siempre aportará alguna información al valor de equiparación de la entidad en la clase.

Se ha utilizado como factor corrector el Nº de arcos + 1, frente a otras medidas basadas en el Nº de hermanos de cada nivel, porque la primera degrada paulatinamente el peso de la propiedad heredada mientras que las segundas lo disminuyen drásticamente.

Por ejemplo, en la BC de la figura 5.18, la propiedad *Esqueleto* es una propiedad específica, es decir, altamente representativa, de la clase *Vertebrado*. Sin embargo, esta propiedad apenas es representativa si se define en los marcos: *Perro*, *Gato*, *Rana*, *Sapo*, etc., pues no se podría utilizar para discriminar individuos que pertenecen a una clase u otra.

2.2) Si la certeza de la propiedad en la nueva clase MC_2 es negativa, entonces, el peso de la propiedad en el marco clase MC_1 se calcularía como:

$$Peso_P = Certeza_P * Representatividad_P \quad (3)$$

En este caso, *Pesop* toma valores en el intervalo [-1, 0] y no es necesario aplicar ningún factor corrector porque si éste se aplicara, se estaría aumentando el peso de la propiedad P al aproximar éste al valor cero.

Dado que $Peso_P$ toma valores en el intervalo cerrado [-1, 1], el significado de este parámetro en el Modelo de Diseño es el siguiente:

1. Si el peso de la propiedad P en el marco clase es uno, entonces, se sabe que los valores almacenados en la propiedad P de la entidad coinciden con los valores de dicha propiedad en la clase y, que la propiedad P es una propiedad específica de la clase. Por ello, la compatibilidad de la propiedad de la entidad E con la propiedad específica de la clase, es máxima. Por ejemplo, si se conoce que una entidad E tiene *Esqueleto* (representatividad 1 en *Vertebrado*) con certeza 1, entonces, la compatibilidad de esta propiedad en la entidad con la propiedad del marco clase *Vertebrado* es absoluta. La técnica de equiparación que se propone no necesitaría más información para determinar que se trata de un *Vertebrado*. Obsérvese que un ser humano, con la misma información, realizaría la misma clasificación. Es importante notar que la representatividad permite al SBM realizar inferencias correctas con información incompleta, al igual que lo hace el humano. Por este motivo, se debe advertir al IC de los peligros derivados de asignar representatividad 1 a las propiedades. De todas formas, no sería complejo introducir en el Modelo familias de propiedades con representatividad 1 para decidir el ajuste.

2. Si el peso de la propiedad P en el marco clase es -1, entonces, se sabe que los valores almacenados en la propiedad P de la entidad, no coinciden con los valores de la propiedad en la clase y, además, la propiedad P es una propiedad específica de la clase. Por ejemplo, si se sabe que una entidad *no* tiene *esqueleto*, entonces, se sabe con total seguridad que la

entidad no pertenece a la clase *Vertebrado*. En estas circustancias un humano realizaría las mismas conclusiones.

3. Si el peso de la propiedad P en el marco clase es cero, entonces, la propiedad P no influye en el proceso de equiparación de la entidad con el marco clase en cuestión.

4. Si el peso de la propiedad P en el marco clase está comprendido en el intervalo abierto (0, 1), entonces la propiedad influye positivamente en el cálculo del valor de equiparación de la entidad con la clase.

5. Si el peso de la propiedad P en el marco clase está comprendido en el intervalo abierto (-1, 0), entonces la propiedad P de la entidad influye negativamente en el cálculo del valor de equiparación de la entidad con la clase.

La tabla 5.3, muestra, para cada marco clase con el que se puede equiparar la entidad, el peso asociado a cada propiedad localizada en el marco u obtenida aplicando herencia (denotado por H) y el criterio con el cual se ha calculado. Se observa que el procedimiento propuesto trabaja con excepciones perfectamente y, por consiguiente, se puede utilizar en sistemas no monótonos. Por ejemplo, en la equiparación de la entidad con el marco *Tortuga Marina* y *Tortuga Terrrestre* el peso asociado a la propiedad *ambiente* se calcula utilizando el valor local almacenado en estos marcos clase y no el valor heredado de la clase *Reptiles*. En la columna *Criterio* se indica la fórmula que se ha utilizado en el cálculo del Peso.

C) Cálculo del Valor de Equiparación

Habiendo calculado los pesos de cada una de las propiedades de cada marco clase, por fin, se procede a calcular el VE de la entidad E en cada marco clase. En este trabajo se utiliza en el cálculo del VE las fórmulas propuestas en MYCIN [Buchanan et al., 84] porque:

a) Garantizan que la combinación de pesos nunca sobrepasa el valor 1 y -1, y, por consiguiente, el VE se mantiene en el intervalor [-1, 1].

b) Si el peso de una propiedad en una entidad E toma el valor 1 (certeza 1, representatividad 1), la entidad es totalmente compatible con la clase y no es necesario conocer más propiedades de la entidad que corroboren la

equiparación. En este sentido, las fórmulas de MYCIN trabajan perfectamente con el concepto de representatividad máxima y certeza máxima.

c) Detecta inconsistencias en caso de tener en un mismo marco clase dos o más propiedades representativas con peso 1 y -1.

En el cálculo del VE pueden darse tres situaciones:

1. Si los pesos p_i y p_j de las propiedades del marco clase son positivos la propagación se realiza utilizando la fórmula:

$$p_i + p_j - p_i * p_j$$

2. Si los pesos son negativos:

$$-(p_i + p_j - p_i * p_j)$$

3. Si uno es positivo y el otro es negativo:

$$\frac{p_i + p_j}{1 - \min\{|p_i|, |p_j|\}}$$

El resultado final de combinar todos los pesos positivos entre sí, los negativos entre sí, y, por último, el peso positivo resultante con el peso negativo según las tres fórmulas anteriores, es el VE de la entidad con el marco Clase.

La tabla 5.4, muestra los valores de equiparación obtenidos al equiparar la entidad en cada clase:

Abreviadamente, el proceso del cálculo del VE se ha mostrado en la figura 5.22. Conocida la certeza positiva o negativa de unos valores de unas propiedades en una entidad E, se transforman estas certezas de propiedades de la entidad en certezas de propiedades de la clase siguiendo las reglas de transformación expuestas en el apartado A. Posteriormente, para cada propiedad con certeza conocida y, para todas aquellas que son accesibles aplicando herencia desde el marco actual, se calcula el *Peso* de la propiedad según las fórmulas descritas en el apartado B. Por último, para cada marco candidato se calcula el VE según las fórmulas y descritas en el apartado C.

Marco Clase	Propiedad	Valor	Certeza	Repres.	Peso	Criteri
Vertebrado	*Esqueleto:*	Si	1	1	1	(1)
Anfibio	*Esqueleto: (H)*	Si	1	1	0,5	(2)
	Respiración:	Pulmonar, Bronquial, Cutánea, Cav. Bucal	0,8	0,1	0,08	(1)
	Ambiente:	Marino, Terrestre	- 1	0,5	-0,5	(1)
Reptil	*Esqueleto: (H)*	Si	1	1	0,5	(2)
	Respiración:	Pulmonar	0,8	0,3	0,24	(1)
	Ambiente:	Marino, Terrestre	- 1	0,2	-0,2	(1)
Ave	*Esqueleto: (H)*	Si	1	1	0,5	(2)
	Respiración:	Pulmonar	0,8	0,2	0,16	(1)
	Ambiente:	Terrestre	0,2	0,3	0,06	(1)
	Nº de Patas:	2	- 1	0,8	-0,8	(1)
Mamífero	*Esqueleto: (H)*	Si	1	1	0,5	(2)
	Mama:	Si	1	1	1	(1)
	Respiración:	Pulmonar	0,8	0,3	0,24	(1)
	Ambiente:	Terrestre	0,2	0,2	0,04	(1)
Rana	*Esqueleto: (H)*	Si	1	1	0,333	(2)
	Respiración: (H)	Pulmonar, Bronquial, Cutánea, Cav. Bucal	0,8	0,1	0,04	(2)
	Ambiente:	Marino, Terrestre	- 1	0,5	-0,5	(3)
	Nº de Patas:	4	1	0,3	0,3	(1)
Sapo	*Esqueleto: (H)*	Si	1	1	0,333	(2)
	Respiración: (H)	Pulmonar, Bronquial, Cutánea, Cav. Bucal	0,8	0,1	0,04	(2)
	Ambiente:	Marino, Terrestre	- 1	0,5	-0,5	(3)
	Nº de Patas:	4	1	0,3	0,3	(1)
Tortuga	*Esqueleto: (H)*	Si	1	1	0,333	(2)
	Respiración: (H)	Pulmonar	0,8	0,3	0,12	(2)
	Ambiente: (H)	Marino, Terrestre	- 1	0,2	-0,2	(3)
	Nº de Patas:	4	1	0,3	0,3	(1)
Tortuga Terrestre	*Esqueleto: (H)*	Si	1	1	0,25	(2)
	Respiración: (H)	Pulmonar	0,8	0,3	0,08	(2)
	Nº de Patas: (H)	4	1	0,3	0,15	(2)
	Ambiente:	Terrestre	0,2	0,5	0,1	(1)

Tabla 5.3. Cálculo de Pesos (Continúa)

Marco Clase	Propiedad	Valor	Certeza	Repres.	Peso	Criterio
Tortuga Marina	*Esqueleto: (H)*	Si	1	1	0,25	(2)
	Respiración: (H)	Pulmonar	0,8	0,3	0,08	(2)
	Nº de Patas: (H)	4	1	0,3	0,15	(2)
	Ambiente:	Marino	-1	0,5	-0,5	(1)
Serpiente	*Esqueleto: (H)*	Si	1	1	0,333	(2)
	Respiración: (H)	Pulmonar	0,8	0,3	0,12	(2)
	Ambiente: (H)	Marino, Terrestre	-1	0,2	-0,2	(3)
	Nº de Patas:	0	-1	0,9	-0,9	(1)
Cetáceo	*Esqueleto: (H)*	Si	1	1	0,333	(2)
	Mama: (H)	Si	1	1	0,5	(2)
	Respiración: (H)	Pulmonar	0,8	0,3	0,12	(2)
	Nº de Patas:	0	-1	0,9	-0,9	(1)
	Ambiente:	Marino	-1	1	-1	(1)
Félido	*Esqueleto: (H)*	Si	1	1	0,333	(2)
	Mama: (H)	Si	1	1	0,5	(2)
	Respiración: (H)	Pulmonar	0,8	0,3	0,12	(2)
	Ambiente: (H)	Terrestre	0,2	0,2	0,04	(2)
	Nº de Patas:	4	1	0,3	0,3	(1)
Cánido	*Esqueleto: (H)*	Si	1	1	0,333	(2)
	Mama: (H)	Si	1	1	0,5	(2)
	Respiración: (H)	Pulmonar	0,8	0,3	0,12	(2)
	Ambiente: (H)	Terrestre	0,2	0,2	0,04	(2)
	Nº de Patas:	4	1	0,3	0,3	(1)
Gato	*Esqueleto: (H)*	Si	1	1	0,25	(2)
	Mama: (H)	Si	1	1	0,333	(2)
	Respiración: (H)	Pulmonar	0,8	0,3	0,08	(2)
	Nº de Patas: (H)	4	1	0,3	0,15	(2)
	Ambiente: (H)	Terrestre	0,2	0,2	0,04	(2)
	Raza:	[Siamés, ...]	0,2	0,9	0,18	(1)
Perro	*Esqueleto: (H)*	Si	1	1	0,25	(2)
	Mama: (H)	Si	1	1	0,333	(2)
	Respiración: (H)	Pulmonar	0,8	0,3	0,08	(2)
	Nº de Patas: (H)	4	1	0,3	0,15	(2)
	Ambiente: (H)	Terrestre	0,2	0,2	0,04	(2)
	Raza:	[Pekinés, ...]	-0,5	0,9	-0,45	(1)

Tabla 5.3. Cálculo de Pesos

	Entidad		
	Propiedad	**Valor**	**Certeza**
	Esqueleto:	Si	1
	Mama:	Si	1
	Respiración:	Pulmonar	0,8
	Nº de Patas:	4	1
	Ambiente:	Marino	-1
	Raza:	Pekinés	-0,5

Marco Clase	Valor de Equiparación
Vertebrado	1
Anfibio	0,08
Reptil	0,525
Ave	-0,4934
Mamífero	1
Rana	0,1036
Sapo	0,1036
Tortuga	0,4863
Tortuga Terrestre	0,4722
Tortuga Marina	-0,1475
Serpiente	-0,8637
Cetáceo	-1
Félido	0,8028
Cánido	0,8028
Gato	0,6921
Perro	0,3173

Tabla 5.4. Valores de Equiparación

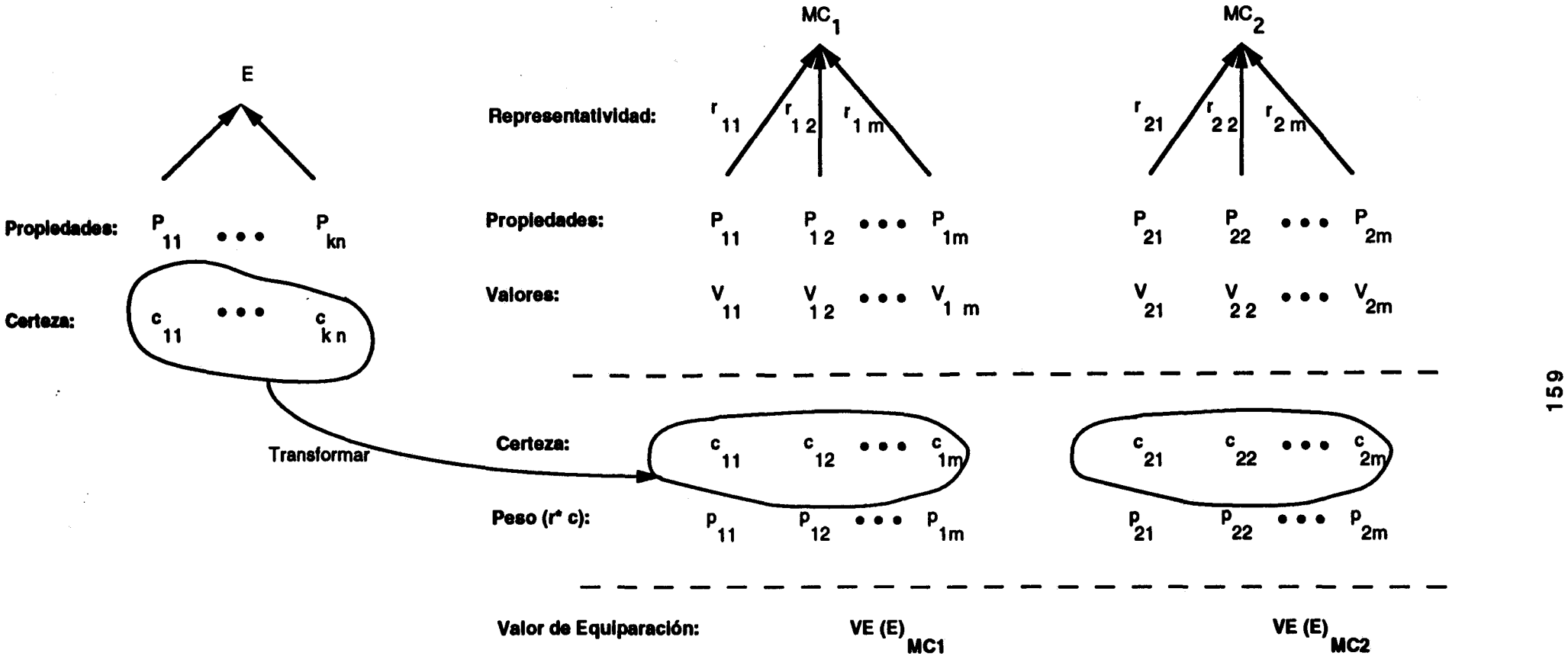


Figura 5.22. Cálculo del Valor de Equiparación

5.3.1.4 Decisión

Calculados los valores de equiparación de la entidad con los marcos clase seleccionados, es el momento de determinar a qué clase pertenece la entidad. Para ello, además del VE calculado, se va a utilizar las relaciones: Clase y Disjunto, definidas por el IC al construir la BC de la aplicación.

Si la unión de los conceptos representados por los i Marcos Clase forman el concepto que ha originado la clasificación, entoces, se sabe que dada una instancia o elemento de éste último, dicho elemento es instancia, como mínimo, de alguno de los marcos del nivel inmediatamente inferior. Si, además, los Marcos MCi son disjuntos se sabe que el elemento será instancia de un único marco. En la figura 5.23, se muestran todas las posibilidades que se pueden dar al dividir una clase en subclases.

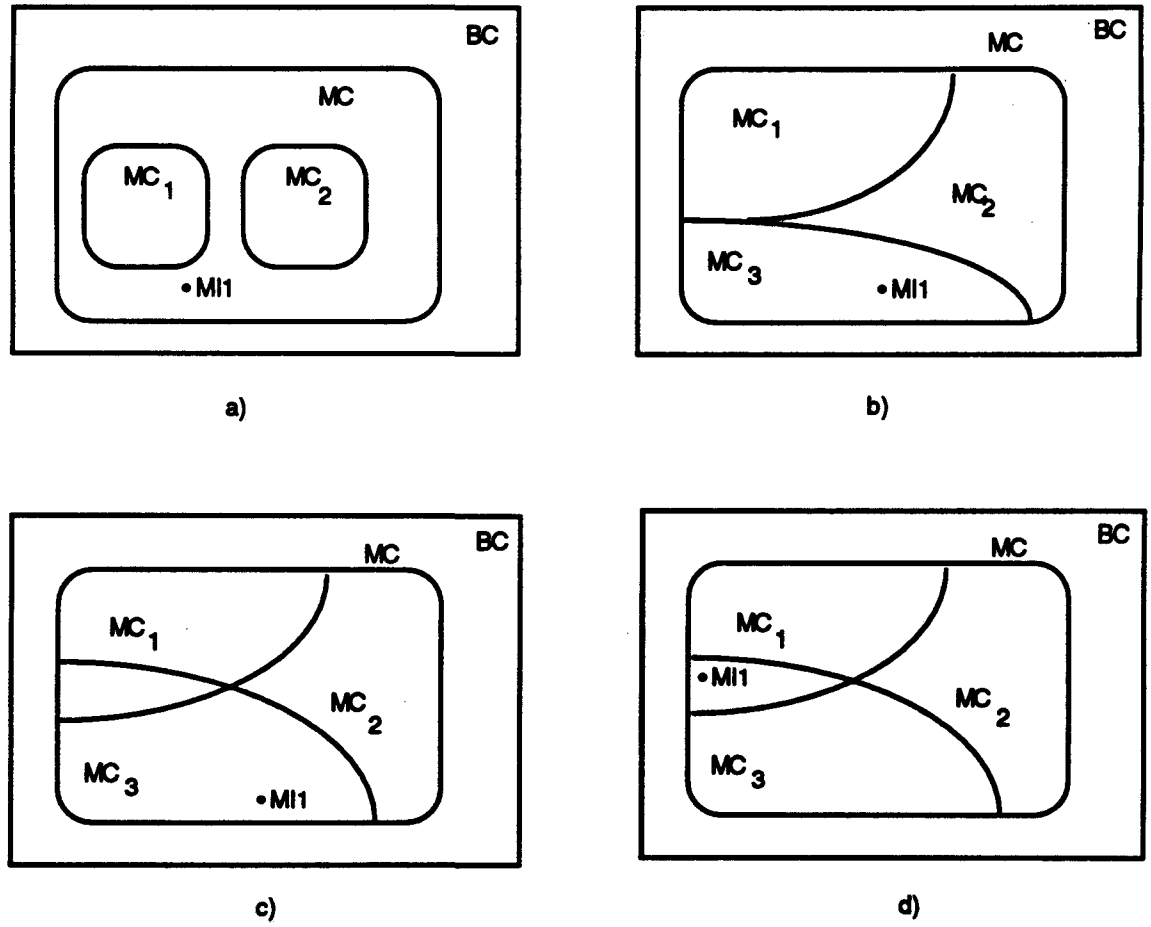


Figura 5.23. Descomposiciones de Clases

Con este fin, se proponen las siguientes reglas, que irían seleccionando aquellos marcos con valor de equiparación más alto:

Regla 1. Se Rechazan los Valores de Equiparación Negativos y con valor cero.

Se rechazan los marcos clases que hayan obtenido un VE negativo o cero. Un VE negativo o cero informa de una inconsistencia en la equiparación de la entidad E con el/los marco(s) clase con los que se intenta la equiparación.

Regla 2. Si existen uno o varios marcos clase con VE = 1, entonces, se rechazan todos los marcos que no sean especializaciones de ellos.

Dado un conjunto de propiedades conocidas con total certeza en una entidad E, si estas propiedades son específicas (representatividad 1) de uno o varios marcos clase, se sabe que la equiparación de la entidad con la clase es totalmente consistente.

Regla 2.1 Especificidad frente a Generalidad

Dados varios marcos clase con VE = 1, se seleccionarán los marcos más específicos frente a lo marcos más generales siempre que éste último se haya especializado en el primero.

Regla 3. Marcos Disjuntos

La relación disjunto se utiliza en la técnica de equiparación para acotar la clase a la cual pertenece la entidad. Dados dos marcos A y B, marcos disjuntos de una jerarquía:

Regla 3.1 $VE(E)_A = VE\ (E)_B$

Se rechazan ambos marcos clase, pues el error que se cometería al equiparar la entidad con ambos marcos clase sería el mismo.

Regla 3.2 $VE(E)_A - VE\ (E)_B > 0{,}2$

En este caso, se seleccionaría el marco clase A frente al B.

Regla 4. Especificidad frente a Generalidad

Dados dos marcos clase A y B, tal que A está incluido en B, y los VE de una entidad E en ambas clases, pueden darse dos situaciones:

Regla 4.1 $VE(E)_A >= VE\ (E)_B$

Se selecciona la clase A frente a la clase B. La equiparación de la entidad E con la clase A frente a la clase B, aporta más consistencia al ser la primera una clase más específica.

Regla 4.2 $VE(E)_B - VE\ (E)_A < 0.2$

Si la diferencia entre el VE en B y en A es menor que 0,2 entonces, al estar A incluido en B se selecciona A frente a B. Pero, si supera este valor, entonces se selecciona B frente a A.

5.3.2. CESION DE PROPIEDADES

Otra técnica de inferencia en Marcos se basa en la transferencia de propiedades y tiene su fundamento en la organización jerárquica o taxonómica de la información. El Modelo de Diseño que aquí se expone propone dos tipos de transferencia o cesión de propiedades: Herencia y Donación.

1. Si en la transferencia de propiedades intervienen marcos padres e hijos conectados por relaciones SubClase e Instancia, se tiene la cesión clásica de propiedades basada en **Herencia.**

2. Si en la transferencia de propiedades no intervienen marcos conectados por relaciones del tipo SubClase e Instancia, se tiene una nueva cesión de propiedades llamada **Donación.**

Ambas, Herencia y Donación, tienen en común el permitir, a los marcos instanciados de una BC, compartir las propiedades definidas y, o, almacenadas en otros marcos de la BC. Las propiedades que se comparten en la herencia y en la donación, dependen, en gran medida, de la semántica asociada a las relaciones que unen el marco que recibe la información con el marco que la cede y, de la distribución que se ha hecho

de las propiedades en la taxonomía de marcos. Sin embargo, la transferencia se realiza de forma diferente en ambos casos:

1. Si la cesión de propiedades se basa en Herencia, la transferencia se realiza desde los marcos clase hacia los marcos instanciados.

2. Pero, si la cesión de propiedades se basa en Donación, la transferencia de los valores de las propiedades se realiza desde marcos instanciados hacia marcos instanciados.

Se pasa a analizar de forma intuitiva las diferencias entre estos dos tipos de cesiones de propiedades. Posteriormente, se analizarán ambas formalmente.

5.3.2.1 Aproximación Intuitiva a la Herencia Y Donación

Este apartado tiene como finalidad mostrar la necesidad de definir unos procedimientos, independientes del dominio, asociados a las relaciones que permitan realizar inferencias con ellas. La ausencia de estos procedimientos obliga a utilizar las relaciones como meras propiedades, desaprovechando una de las funcionalidades más interesante del formalismo de Marcos: la inferencia basada en relaciones entre conceptos.

Las relaciones y las propiedades tienen en común que ambas se representan declarativamente en los marcos como ranuras. Sin embargo, esto no significa en absoluto que sean lo mismo. Por ejemplo, supóngase la jerarquía de marcos representada en la figura 5.24.

Las respuestas que un ser humano y un SBC darían al realizar las siguientes consultas para la instancia I_1 son:

Consulta 1: *> Valor (I1, Nombre)*
SBC: *> Pepe*
Humano: *> Pepe*

Consulta 2: *> Valor (I1, Casado_Con)*
SBC: *> Clara*
Humano: *> Clara*

Consulta 3: > *Valor (I1, Instancia)*

SBC: > *Hombre*

Humano: > *Hombre*

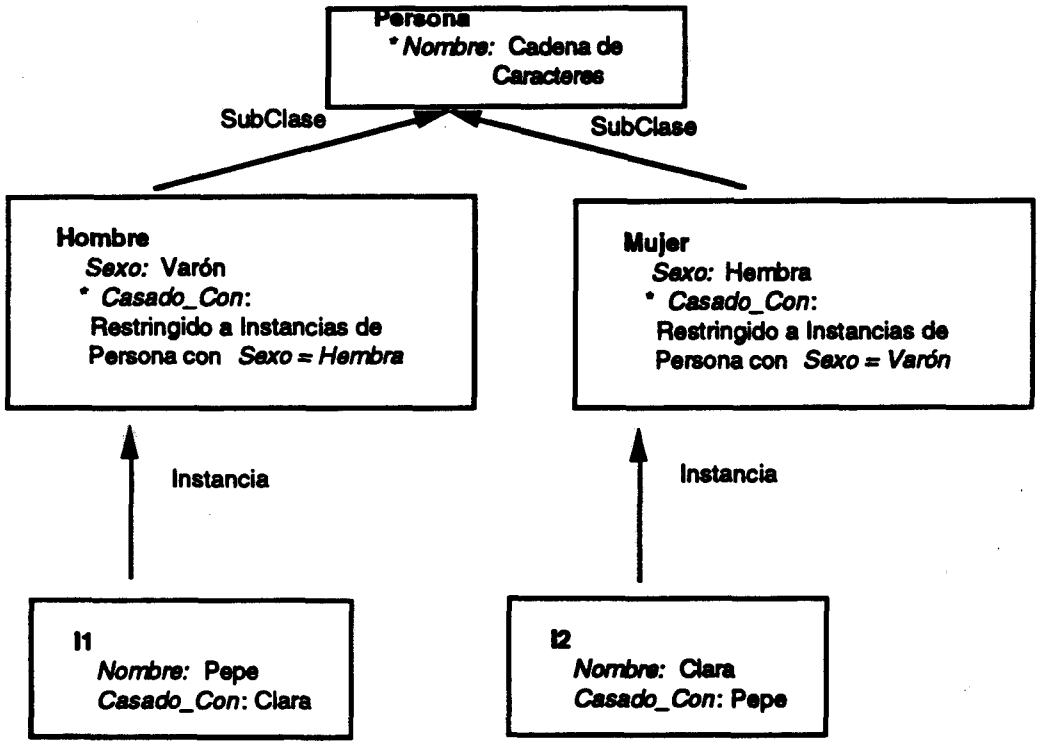


Figura 5.24. La ranura *Casado_Con* es una propiedad

Obsérvese, que existe una diferencia fundamental entre estas consultas: en las dos primeras, se pregunta por valores de ranuras que representan propiedades, mientras que en la tercera consulta, se pregunta por valores de ranuras que representan relaciones. En cualquiera de los tres casos, las ranuras se han utilizado de forma declarativa y no para realizar inferencias. Otra consulta declarativa sobre ranuras que son relaciones sería:

Consulta 4: > *Valor (Mujer, SubClase)*

SBC: > *Persona*

Humano: > *Persona*

Un ejemplo en el que una relación se utiliza procedimentalmente, es el siguiente:

Consulta 5: > *Valor (I1, Sexo)*

SBC: > *Varón*

Humano: > *Varón*

Esta última consulta obliga al SBM a utilizar la ranura *Instancia* procedimentalmente pues, al no encontrarse la propiedad buscada en el marco instanciadado I1, se busca la propiedad en marcos superiores conectados con el marco instanciado actual.

Supóngase ahora que se modifica ligeramente la jerarquía de la figura 5.24 en la jerarquía de la figura 5.25, en la que la propiedad *Casado_Con* se representa como una relación entre los marcos *Hombre* y *Mujer*. Si, sobre esta jerarquía, se realizan las consultas anteriores, las respuestas que se obtienen son las mismas.

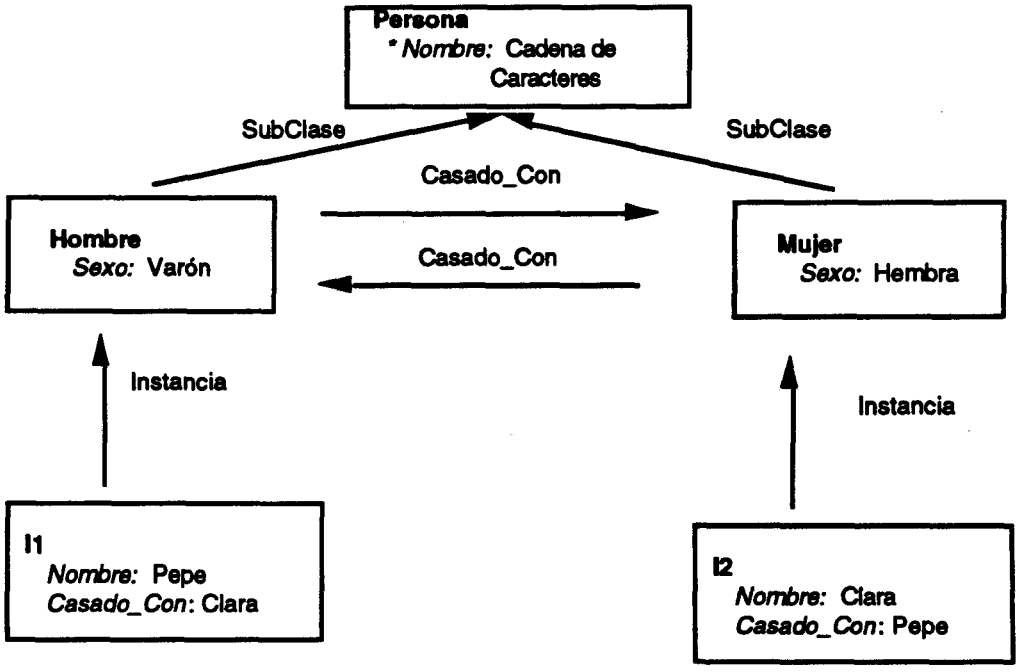


Figura 5.25. La ranura *Casado Con* es una Relación.

Si una propiedad se puede representar como una relación, cabe planterse cual de las dos representaciones es más ventajosa. Existen dos posibles respuestas:

1. Si lo que interesa es que el sistema proporcione las respuestas adecuadas, entonces, se puede decir que las dos representaciones son funcionalmente equivalentes al proporcionar las mismas respuestas. Pero, las dos representaciones no son equivalentes computacionalmente pues la definición dada en la figura 5.24 es más compleja que la de la figura 5.25. Esto se debe a que en la jerarquía de la figura 5.24, la semántica de la propiedad *Casado_Con* está sujeta a unas restricciones que deben ser explícitamente representadas en la BC, mientras que en la figura 5.25 las restricciones vienen dadas implícitamente por los

marcos que conecta la relación, siendo imposible aplicar la relación *Casado_Con* entre marcos que no sean instancias de *Hombre* y *Mujer*.

2. Al no existir ningún procedimiento asociado a la semántica de la relación *Casado_Con* en la jerarquía de la figura 5.25, es imposible realizar inferencias con ella y, por consiguiente, ambas representaciones son equivalentes. Sin embargo, si se quisiera razonar sobre esta propiedad, sería más ventajoso el uso de relaciones.

Antes de proponer una técnica que realice inferencias sobre relaciones ad hoc, se van a analizar las inferencias asociadas a la relación SubClase. Estas inferencias vienen determinadas por las propiedades y por la semántica de la relación, y son las inferencias clásicas utilizadas en los Marcos.

Se sabe, por el estudio realizado en el apartado 5.2.2.1, que la relación SubClase no posee la propiedad simétrica pero sí la propiedad transitiva. La semántica asociada a la transitividad de la relación SubClase es la siguiente: Si la clase A está incluida en la clase B y la clase B está incluida en la clase C, entonces, la clase A está incluida en la clase C. Esta relación de inclusión captura el conocimiento de sentido común existente en el mundo real, por el cual, clases que están incluidas en otras clases o instancias de la clase tienen las mismas características, precisamente, por pertenecer a una misma clase. Al procedimiento que utiliza, en un SBM, la transitividad de la relación SubClase para realizar inferencias se le llamó *Herencia de Propiedades*. Por tanto, los procedimientos que implementan la Herencia están capturando el conocimiento de sentido común, por el cual, marcos clase acceden a todas las propiedades definidas en los marcos clase a que pertenecen. Dada la semántica de la relación SubClase, el tipo de inferencia realizado se traduce en que todas las propiedades de marcos clase son exportables o heredables por marcos de niveles inferiores en la relación SubClase.

Supóngase, ahora, que se está construyendo un SBC que gestiona los recursos de la red de computadoras de la figura 5.26, donde C_1, C_2, C_3, C_4 y C_5 son computadoras, I_1 e I_2 impresoras y, F_1, F_2, F_3, F_4 y F_5 y F_6 ficheros. Supóngase que el SBM con los tipos de computadoras que configuran la red es el de la figura 5.27. En el SBM, cada computadora de la red es una instancia de un tipo de computadora y rellenará, mediante herencia, las propiedades instancia definidas en las clases con las que está unido. Supóngase, también, que en el marco raíz de la jerarquía de computadoras, se ha

definido una relación llamada *Conexión* que tiene como origen y destino el marco clase raíz de la jerarquía de computadoras.

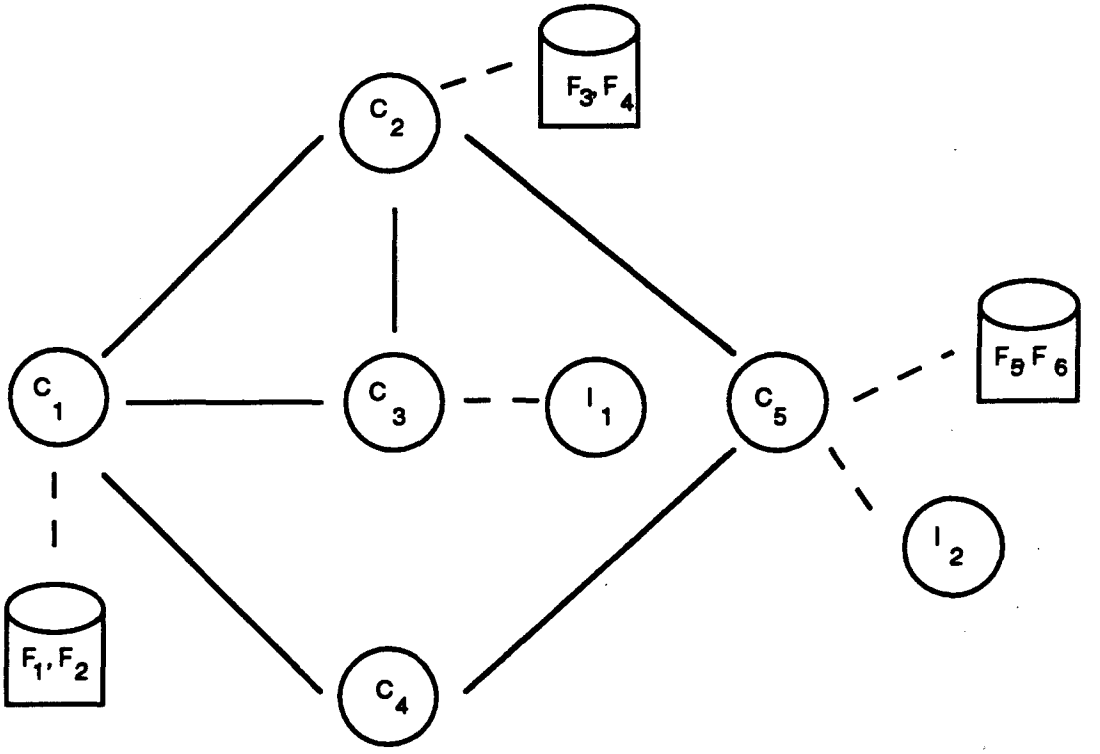


Figura 5.26. Red de Computadoras

Si a un ser humano y al SBM se plantean las siguientes consultas, las respuestas que se obtienen, suponiendo todos las computadoras conectadas y funcionando, son:

Consulta 6: > *Valor(Computadora3, Velocidad)*

SBC:	> 32 MHz	(Local)
Humano:	> 32 MHz	(Local)
	> 32 MHz	(En Red)

Consulta 7: > *Valor(Computadora3, Periféricos)*

SBC:	> Impresora1	(Local)
Humano:	> Impresora1	(Local)
	> Impresora1, Impresora2	(En Red)

Consulta 8: > *Valor(Computadora3, Ficheros)*

SBC:	> Ninguno	(Local)
Humano:	> Ninguno	(Local)
	> F_1, F_2, F_3, F_4, F_5, F_6	(En Red)

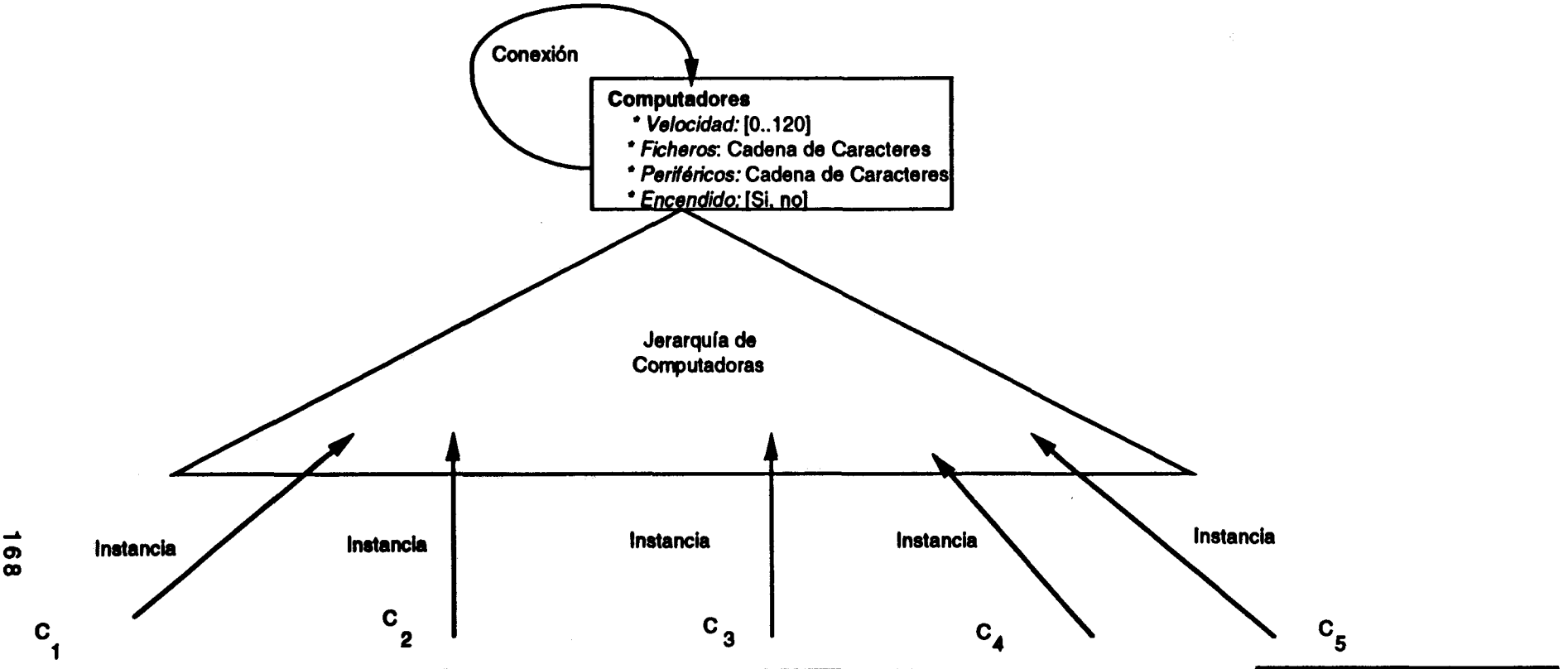


Figura 5.27. SBM para la red de Computadoras

Consulta 9: > *Valor(Computadora3, Conexión)*

SBC: > Computadora1, Computadora2

Humano: > Computadora1, Computadora2, Computadora4, Computadora5

Obsérvese que las respuestas del humano y del SBC coinciden si cada computadora se considera aislada en la red y no conectada a otras computadoras. Una primera aproximación consistiría en definir en la jerarquía de marcos las propiedades *Periféricos_Accesibles* y *Ficheros_Accesibles*, y rellenar, en cada marco instanciado, estas ranuras. Esta solución, aunque es correcta, presenta dos objeciones. La primera consiste en que el IC debe rellenar todas las propiedades en cada marco instanciado, y, la segunda, y no por ello menos importante, se encuentra en que las diferentes respuestas dadas por el humano y por el SBM se deben a que el humano, basándose en su sentido común, utiliza la relación *Conexión* para realizar inferencias y el SBM no. El humano sabe que si la *computadora* C3 está conectada con la *computadora* C2 y ésta con la C5, entonces, aplicando la transitividad de la relación *Conexión*, la *computadora* C3 está conectada con la C5 y, por consiguiente, podrá utilizar los *Periféricos y Ficheros* en ella definidos. Si se quiere que el SBM realice estas inferencias, no basta con introducir la relación *Conexión* como una ranura declarativa, es necesario dotar a la relación *Conexión* de un conjunto de procedimientos que capturen el conocimiento de sentido común, y permitan al SBC realizar las mismas inferencias que puede realizar el humano. Es decir, utilizar la transitividad de la relación *Conexión* para acceder a los valores almacenados en las propiedades *Periféricos* y *Ficheros*, pero no a los valores de la propiedad *Velocidad*. A la técnica de inferencia que utiliza las relaciones "ad hoc" para realizar cesión de propiedades se la llama en el Modelo de Diseño que se propone **Donación.**

5.3.2.2 Relaciones de Herencia y Donación

Los procedimientos que realizan la transferencia de propiedades comienzan a ejecutarse cuando el IC plantea una consulta al sistema. Básicamente, las consultas planteadas a cualquier SBM serán de dos tipos:

a) *Valor (MI, P)*, si se desea conocer el valor de la propiedad P en el marco instanciado MI. En este caso, el sistema realizaría cesión de propiedades basada en Herencia y, opcionalmente, Donación.

b) *Valor (MI, P, R)*, si se desea conocer el valor de la propiedad P en el marco instanciado MI basándose en la relación R, el sistema realizaría cesión de propiedades basada en Donación y, opcionalmente, Herencia.

Ambas técnicas, Herencia y Donación, se han relacionado como se muestra en la figura 5.28 con el fin de incrementar la cesión de propiedades y acercar las respuestas del SBM a las respuestas de un humano inteligente.

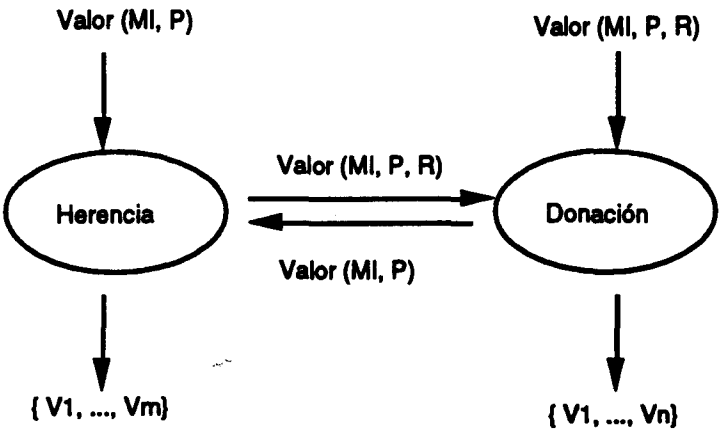


Figura 5.28. Relaciones de Herencia y Donación.

La figura 5.28 indica que:

1. *Valor (MI, P)*

   Si el usuario desea conocer el valor de la propiedad P en el marco instanciado MI aplicando herencia y no encuentra ningún valor (bien porque el valor que toma la propiedad es desconocido, bien porque la propiedad toma el valor "Ninguno"), entonces, el sistema buscará si se ha conectado el marco instanciado mediante alguna relación "ad hoc" con otro marco instanciado y, además, en dicha relación se ha definido P como propiedad exportable. Si esto ocurre, la técnica de herencia llamará a la técnica de donación solicitando la ejecución de: *Valor (MI, P, R)*. Si existieran varias relaciones a medida que tuvieran a P como propiedad exportable, por omisión, si el IC no detecta la situación o no define preferencias, se devuelve la unión de todos los valores accesibles desde todas las relaciones.

   Por ejemplo, tomando como referencia la BC de la red de computadoras de la figura 5.26, si al sistema se realizaran las consultas siguientes las respuestas obtenidas serían:

Consulta 10: > *Valor(Computadora2, Ficheros)*
SBC: > F_3, F_4

Consulta 11: > *Valor(Computadora3, Ficheros)*
SBC: > F_1, F_2, F_3, F_4, F_5, F_6

Obsérvese que en la consulta 10, al encontrarse almacenados los valores locales de la propiedad *Ficheros* en el marco instanciado, no es necesario aplicar la técnica de donación. Sin embargo, en la consulta 11, al no encontrarse almacenado los valores locales de la propiedad *Ficheros* en el marco instanciado, sí es necesario aplicar la técnica de donación.

2. *Valor (MI, P, R)*

Si el usuario desea conocer el valor de la propiedad P en el MI en la relación "ad hoc" R, entonces, se aplicará cesión de propiedades basada en donación. Pero, si al acceder al valor de la propiedad P en un marco instanciado, el valor local almacenado en la ranura es desconocido, entonces, se aplicará la técnica de herencia para asignar valores locales a la propiedad.

Al plantear a la jerarquía anterior las consultas siguientes, como todas las propiedades han sido rellenadas en las instancias, no ha sido necesario aplicar herencia de propiedades.

Consulta 12: > *Valor(Computadora2, Ficheros, Conexión)*
SBC: > F_1, F_2, F_3, F_4, F_5, F_6

Consulta 13: > *Valor(Computadora3, Ficheros, Conexión)*
SBC: > F_1, F_2, F_3, F_4, F_5, F_6

Por tanto, se puede hablar de una técnica *híbrida* de cesión de propiedades que combina la Herencia con la Donación. Esta cesión de propiedades se representaría en dos dimensiones como se muestra en la figura 5.29.

El algoritmo que refleja los dos conceptos anteriores, algoritmo que, por otro lado, evita la presencia de bucles, sería el mostrado en la figura 5.30.

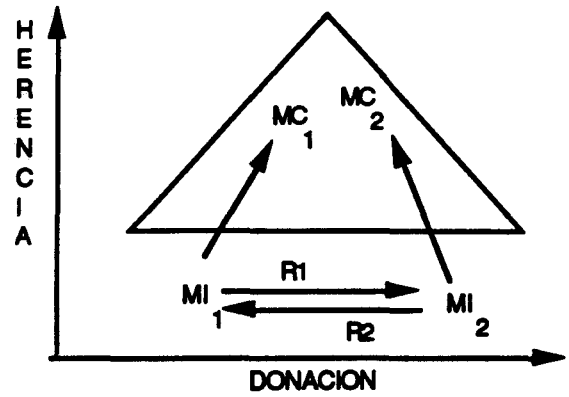


Figura 5.29. Dos dimensiones de la Cesión de propiedades.

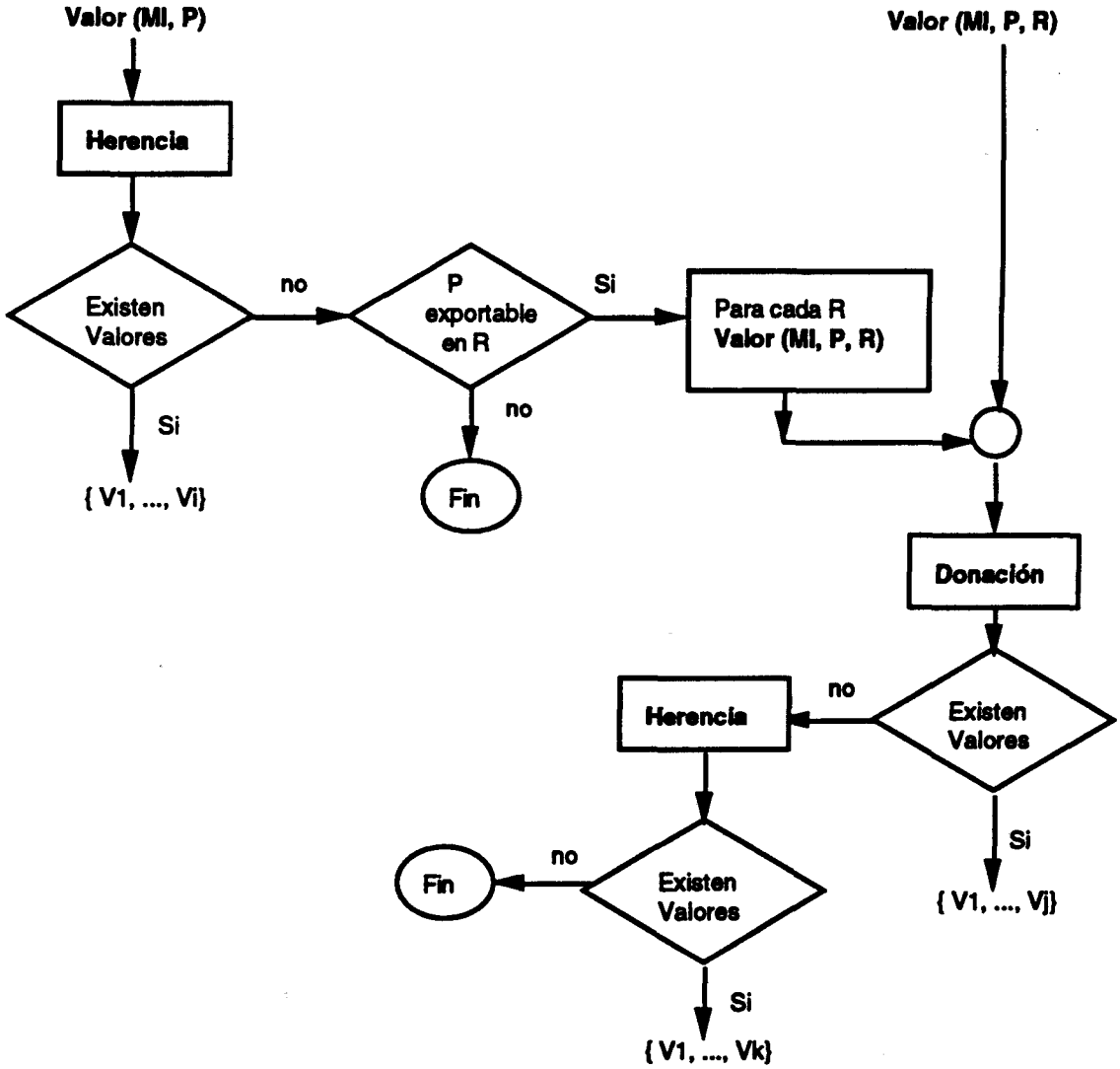


Figura 5.30. Algoritmo híbrido de cesión de propiedades.

5.3.2.3 Herencia de Propiedades

En este apartado se propone un sistema de herencia múltiple, polar, no monótono y homogéneo para un SBM que utiliza los conceptos planteados en la Teoría Matemática formal basada en la distancia "inferencial" de Touretzky [Touretzky, 84]. El sistema de herencia que en este trabajo se propone contempla los siguientes apartados:

1. Dado el conocimiento representado en un grafo acíclico dirigido con enlaces bipolares, se propone la traducción del conocimiento en él representado, a un SBM en el cual únicamente intervienen enlaces polares.

2. Se estudiará la aplicabilidad de los conceptos asociados a los sistemas de herencia no procedimentales sobre grafos acíclicos dirigidos a los SBM.

3. Se propondrá un sistema de herencia múltiple polar, no monótono, homogéneo y procedimental que trabaje con los conceptos de marcos clase y marcos instanciados descritos en el apartado 5.2, y que aconseje, en el caso de detectar ambigüedades, el marco del cual se debe heredar la propiedad ambigua.

5.3.2.3.1. Conversión del Grafo Acíclico Dirigido a Marcos

En este apartado se analiza cómo el conocimiento expresado en un grafo acíclico dirigido se transforma en una jerarquía de marcos. En primer lugar, se va a traducir la nomenclatura expuesta por Touretzky para grafos acíclicos dirigidos a la notación de Marcos, y, en segundo lugar, se traducirán los grafos de Clyde, Nixon y Tweety que son ejemplos de grafos problemáticos, por la presencia de redundancia y ambigüedad, al formalismo de Marcos.

a) Conversión de la Nomenclatura

La transformación de un grafo acíclico dirigido a una jerarquía de marcos implica traducir los nodos y los arcos del grafo de acuerdo a las siguientes reglas:

a.1) Conversión de los nodos.

Se distinguen tres casos:

* Si *a* representa un individuo en un grafo acíclico dirigido, *a* pasa a ser un marco instanciado en los SBM.

* Si *P* representa una clase en un grafo acíclico dirigido, *P* pasa a ser un marco clase en los SBM.

* Si *q* representa una propiedad en un grafo acíclico dirigido, *q* es una propiedad propia en los SBM.

a.2) Conversión de los arcos

Comprende los seis casos que a continuación se citan:

a.2.1) $a \rightarrow P$

Si *a* es un individuo y *P* es una clase, la semántica del arco $a \rightarrow P$ significa que *a* es una instancia de la clase *P*, conocimiento que se expresa en marcos según se indica en la figura 5.31.(a) y en la Teoría de Conjuntos según la figura 5.31.(b).

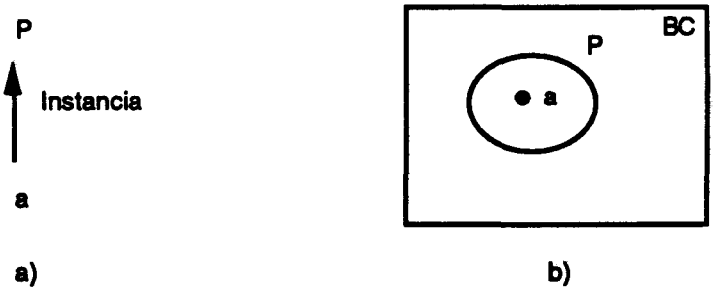


Figura 5.31. Traducción de $a \rightarrow P$

a.2.2) $a \not\rightarrow P$

Si *a* es un individuo y *P* es una clase, la semántica del arco $a \not\rightarrow P$ significa que *a* no es un individuo o instancia de *P*. El Diagrama de Venn de la figura 5.32 expresa estos conceptos.

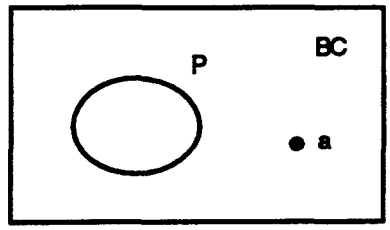


Figura 5.32 Traducción de *a* $\not\rightarrow$ *P*

Los marcos, son sistemas polares en los que sólo aparecen enlaces positivos. El conocimiento de que las instancias de *a* no están en la clase *P* se puede introducir en la BC de varias formas:

2.a Definiendo una relación "ad hoc" llamada *No_Instancia* entre un marco instanciado y un marco clase. Esta nueva relación, si no existen unos procedimientos que la utilicen para realizar inferencias, no proporciona ningún beneficio al SBC que la incluya, y, por consiguiente, no debería introducirse en la BC del sistema.

2.b. Crear una clase *No_P* con todas las propiedades que poseen todos los individuos que, explícitamente, se sabe que no son instancias de *P*. Si *U* es la clase universal y *P* es una clase de *U* , entonces, se puede expresar *No_P* como: *No_P = U - P.*

Al definir la clase *No_P* como el universo menos *P*, *No_P* tendría que almacenar todas las propiedades conocidas de *U* que no están en *P*. Este enfoque se rechaza porque presenta el problema de seleccionar las propiedades más significativas del conjunto de individuos que son instancias de la clase *No_P*, pues pueden existir propiedades definidas en *No_P* que son ciertas para la instancia *a*, y, pueden no serlo para la instancia *b*, instancia que también puede tener un enlace de la forma *b* $\not\rightarrow$ *P*. Gráficamente, la relación entre *No_P* y *P* se muestra en el diagrama de Venn de la figura 5.33.

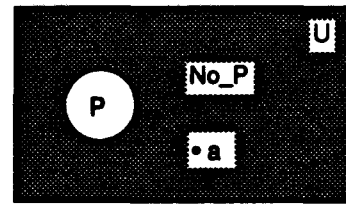


Figura 5.33. *No_P = U - P*

2.c Un enfoque alternativo sería el siguiente. Si se sabe que un individuo *a* no es instancia de la clase *P*, entonces, se sabe que para ese individuo no son ciertas algunas o todas las propiedades de clase definidas en *P*, pues si lo fueran, sería una instancia de *P* y se pasaría a rellenar las propiedades de instancia definidas en *P*.

La solución que se analiza es utilizar una clase *No_P*, en la cual, se van a dar nuevos valores a todas las propiedades de clase definidas en *P*. La definición de propiedades de instancia en *No_P* no garantiza que estas propiedades que representan prototipos tengan sentido en todos y cada uno de las instancias que están unidas mendiante enlaces negativos con *P*. Es decir, las propiedades de instancia definidas en *No_P* pueden describir correcta o incorrectamente a algunos prototipos y dejar a otros sin describir. Por estos motivos, esta solución es inviable y se rechaza. Gráficamente, se muestra en la figura 5.34 cómo se ha acotado *No_P* en *U*.

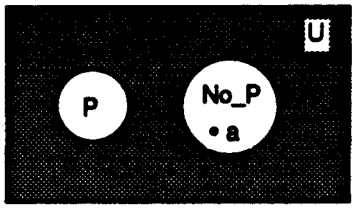


Figura 5.34. *No_P* es una clase en *U*.

2.d No representar el conocimiento de que *a* no es una instancia de la clase *P*. Esta opción no representa el conocimiento y, por consiguiente, impide la realización de inferencias, pues es imposible razonar con conocimiento que explícitamente se ha omitido. Esta última solución ha sido la decisión adoptada en la traducción del enlace a $\not\rightarrow$ P en este trabajo.

a.2.3) $P \rightarrow Q$

Si *P* y *Q* son clases, entonces, el enlace $P \rightarrow Q$ de un grafo acíclico dirigido se representa en una jerarquía de marcos como se muestra en la figura 5.35, pues la clase *P* está incluida en la clase *Q*.

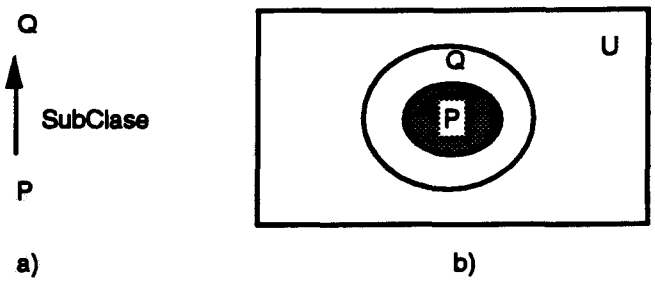


Figura 5.35. $P \rightarrow Q$

a.2.4) $P \not\rightarrow Q$

Si P y Q son clases, la semántica del enlace $P \not\rightarrow Q$ significa que la Clase P no es una especialización de la clase Q en U, o, lo que es lo mismo, que P no está incluida totalmente en Q, dándose únicamente las relaciones de la figura 5.36 entre P y Q.

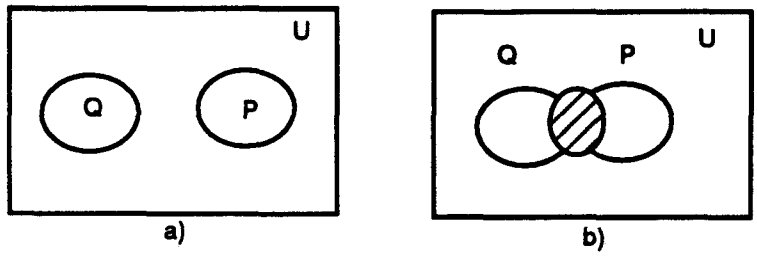


Figura 5.36. $P \not\rightarrow Q$

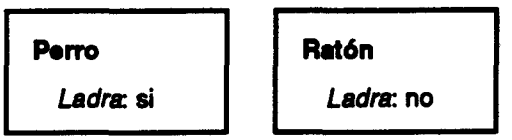
Básicamente, el razonamiento que se realiza en esta situación es similar al razonamiento realizado en $a \not\rightarrow P$. Al igual que allí ocurría, la decisión que se toma es no representar la relación.

a.2.5) $P \rightarrow q$

Si P es una clase y q es una propiedad, la semánica del enlace $P \rightarrow q$ significa que en P se encuentra definida la propiedad q. En un SBM, se definirá la propiedad q como una propiedad de clase del marco P que se rellena con el valor *sí*. Por ejemplo, *Perro* $\rightarrow$ *Ladra* se representaría en un SBM como muestra la figura 5.37.(a).

a.2.6) $P \not\rightarrow q$

Si P es una clase y q una propiedad, la semántica del enlace $P \not\rightarrow q$ significa que P no posee la propiedad q. En un SBM se definirá una propiedad de clase q en P que se rellenará con el valor *no*. Por ejemplo, *Ratón* $\not\rightarrow$ *Ladra* se representaría, en un SBM, como muestra la figura 5.37.(b).



a) b)

Figura 5.37. P $\rightarrow$ q y P $\not\rightarrow$ q

b) Ejemplos de Conversión

La figura 5.38, 5.39 y 5.40, muestran las jerarquías de marcos asociadas a los grafos acíclicos dirigidos de Nixon, Clyde, y Tweety respectivamente que representan los problemas siguientes:

a) La transformación del grafo acíclico dirigido en un SBM no evita la ambigüedad al preguntar en *Nixon* por la propiedad *Pacifista.*

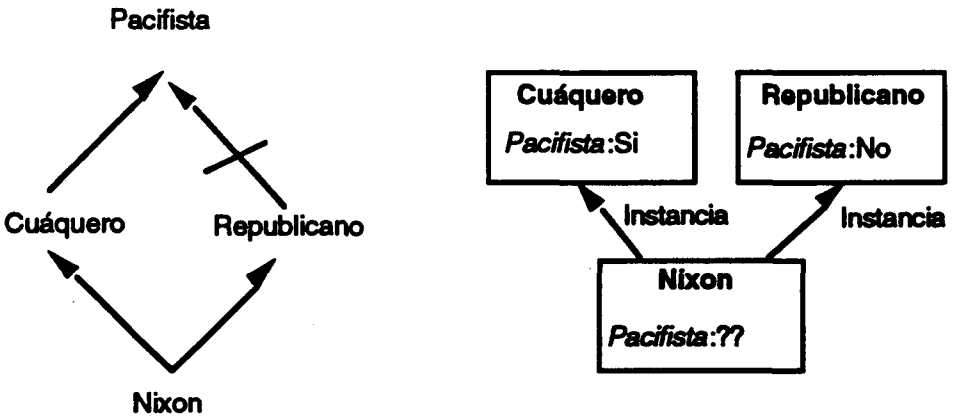


Figura 5.38. Jerarquía de Marcos de Nixon

b) Al transformar el grafo acíclico dirigido en un SBM y aplicar la distancia "inferencial" como técnica de herencia, se observa que se detecta la relación redundante desde *Clyde* hacia *Elefante* y se devuelve el valor almacenado en la clase más específica, frente al valor almacenado en la más general. Para que esto ocurra hay que introducir en el marco

clase *Elefante Real* la propiedad *Color* y rellenarla con el valor *Blanco*. Es decir, hay que añadir conocimiento adicional a la jerarquía.

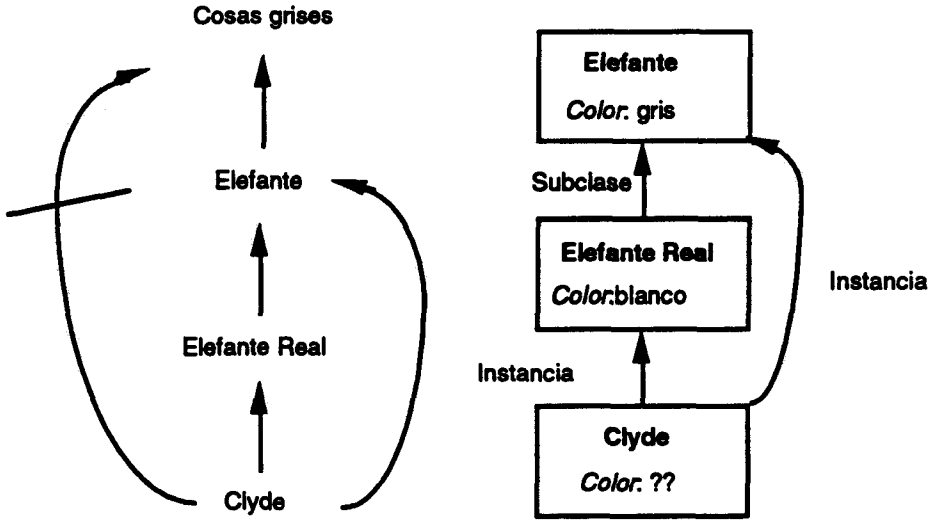


Figura 5.39. Jerarquía de Marcos de Clyde

c) Al transformar el grafo acíclico dirigido en un SBM se observa que el problema de la neutralización de caminos ha desaparecido.

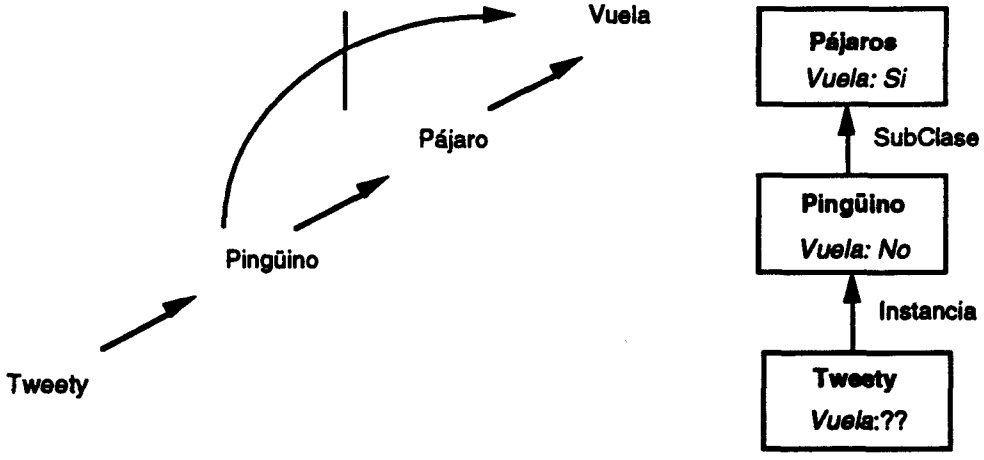


Figura 5.40. Jerarquía de Marcos de Tweety

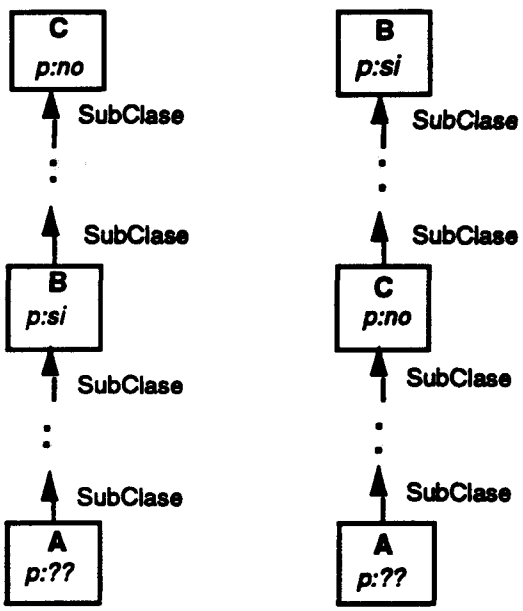
5.3.2.3.2 *La Distancia "Inferencial" en Marcos*

Supóngase dos marcos clase *B* y *C* en los que se han definido las propiedades *p* y $\neg p$, y un marco *A* en el que se pregunta por el valor de la propiedad *p*.

El concepto de distancia "inferencial" en grafos acíclicos dirigidos se traduce en una jerarquía de marcos como, gráficamente, se muestra en la figura 5.41, y se define como:

> *La condición necesaria y suficiente para que la clase A esté más cercana a la clase B que a la clase C, es que la clase A tenga un camino de inferencia a través de la clase B hacia la clase C.*

Nótese que en el concepto de distancia "inferencial" adaptado a los SBM aparecen marcos conectados por enlaces de tipo SubClase. Por este motivo, al no permitir comparar marcos no conectados, el orden introducido por la distancia "inferencial" sigue siendo un orden parcial y, por consiguiente, se pueden heredar valores contradictorios definidos en ramas que no están conectadas.



a) *A* tiene *p*. b) *A* no tiene *p*.

Figura 5.41. Distancia "inferencial" en Marcos.

Al aplicar la distancia "inferencial" a la jerarquía de marcos:

1. Se detecta la presencia de conocimiento redundante, prevaleciendo siempre el conocimiento más específico sobre el conocimiento más general. Al preguntar en la jerarquía 5.39 por el color de *Clyde*, se obtendría que es de color *blanco*.

2. Se detecta la presencia de ambigüedades en la jerarquía de marcos, pero no se resuelve la ambigüedad. En este caso, al preguntar en la jerarquía de la figura 5.38 si *Nixon* es *pacifista*, el sistema de herencia detectaría una ambigüedad, permaneciendo la propiedad *pacifista* sin ningún valor, antes de provocar una contradicción.

3. Detecta la presencia de conocimiento no especializado y devuelve la solución correcta. Como ejemplo, se tiene la jerarquía del avestruz *Fifi* descrita en el Estado de la Cuestión en la figura 2.8. En este caso, la distancia "inferencial" devuelve el valor almacenado en la clase más específica.

4. El problema de las inconsistencias en los caminos directos de un grafo acíclico no dirigido, no aparece en los SBM. Esto se debe a que la definición de cada una de las propiedades de un marco se realiza solamente una única vez, impidiendo que un mismo marco se defina dos veces la misma propiedad de clase con valores diferentes.

5. La neutralización de caminos generales por otros más específicos que llevan a heredar valores contradictorios para una propiedad *p* en un grafo acíclico dirigido no es necesaria en los SBM. En la jerarquía de la figura 5.40, al preguntar si Tweety *vuela*, el sistema respondería que no, debido a que la clase *Pingüino* es más especifica que la clase *Pájaro*.

5.3.2.3.3 Situaciones que rigen la herencia de propiedades

En este apartado, se propone un conjunto de inferencias basadas en herencia de propiedades que utiliza la distancia "inferencial" adaptada a los SBM, y la combina con la técnica de donaciones. Se tiene este tipo de herencia cuando se solicita el valor de una ranura R de un marco instanciado que está unido, por una o varias relaciones Instancia, a uno o varios marcos clase.

Pueden darse cuatro situaciones:

Situación1: *Herencia de Propiedades de Instancia en un Marco Instanciado.*

El marco instanciado rellenará las propiedades de instancia definidas en todas las clases con las que está relacionado directamente mediante relaciones Instancia o a través de relaciones SubClase.

Situación 2: *Herencia de Propiedades de Clase en un Marco Instanciado.*

El marco instanciado accederá a las propiedades de clase definidas en todas las clases con las que está relacionado directamente o a través de relaciones SubClase.

Situación 3: *Llamada a Donación.*

Si se solicita al sistema el valor que toma la propiedad P en un marco instnaciado aplicando herencia, sólamente se utilizará la técnica de donación si se dan alguna de las tres situaciones siguientes:

a) No se encuentra definida la propiedad en el/los camino(s) que une(n) el marco instanciado con el marco raíz de la jerarquía.

b) Se ha encontrado la propiedad en la jerarquía, pero no se ha rellenado con ningún valor por ser estos desconocidos.

c) Se ha encontrado la propiedad en la jerarquía y explícitamente se ha rellenado con el valor "Ninguno".

Entonces, si la propiedad buscada se ha definido como propiedad exportable en las relaciones "ad hoc" que parten del marco instanciado en cuestión, el sistema realizará tantas consultas del tipo *Valor (MI, R, P)* como relaciones "ad hoc" partan del marco instanciado.

Situación 4: *Resolución de Ambigüedades.*

Dado que en las dos primeras situaciones pueden existir ambigüedades, se ha decidido calcular el VE del marco instanciado con los marcos clase en los que se encuentra definida la propiedad para deshacer la ambigüedad. Se heredará la propiedad ambigua del marco clase que proporcione el VE más alto.

5.3.2.3.4.4 Algoritmo de Herencia

Descritas las reglas que rigen la herencia de propiedades en SBM, el algoritmo que permite combinar los dos tipos de herencia, modificación del algoritmo propuesto

por Rich, es el que a continuación se expone y cuyo organigrama se muestra en la figura 5.42. Supóngase que se desea conocer el valor de una ranura R en un marco instanciado MI.

Paso1: Si en el marco instanciado se encuentra la ranura R, se devuelve su valor y se va al Paso 7. En caso contrario, se va al Paso 2.

Paso 2: Inicializar la lista de marcos CANDIDATOS con vacía.

Paso 3: Al no encontrarse la propiedad almacenada en el marco instanciado utilizando las relaciones de instanciación, se accede a los marcos clases con los que está relacionado para ver si en ellos se encuentra la propiedad buscada, y se va al paso 4.

Paso 4: Para cada marco clase, se realiza búsqueda en amplitud o en profundidad. En cada paso, se comprueba si se ha encontrado en el marco clase un valor para la ranura R.

4.1 Si se encuentra el valor, se introduce el marco en la lista de marcos CANDIDATOS y se termina de explorar esta rama.

4.2 Si no se encuentra, utilizando la relación SubClase, se sube un nivel más en la jerarquía.

4.3 Si ya no quedan más ramas por explorar se va al Paso 5.

Paso 5: Para cada elemento C de la lista de marcos CANDIDATOS:

5.1 Se comprueba si en la lista de marcos CANDIDATOS se encuentra un marco clase más cercano al marco instanciado que el marco C, de tal forma que exista un camino de inferencia desde dicho marco hasta C.

5.2 Si existe, entonces se borra C de la lista de marcos CANDIDATOS.

Paso 6: Analizar el número de elementos de la lista de marcos CANDIDATOS

6.1 Si es cero, entonces, en ningún marco clase se ha definido la ranura R. En este caso, si la propiedad buscada es una propiedad definida como una propiedad exportable en una o

varias relaciones R que parten del marco instanciado, entonces, para cada relación se realizará una llamada al algoritmo de Donación. Posteriormente, se va al paso 7.

6.2 Si es uno, entonces, se devuelve el valor que toma la ranura siempre que ésta sea diferente del valor "Ninguno" y se va al paso 7. Pero, si la ranura se ha rellenado con dicho valor o éste es desconocido, entonces, se realiza una llamada al algoritmo de Donación siempre que la propiedad en cuestión sea una propiedad exportable en alguna relación "ad hoc" R que parta del marco instanciado. En caso contrario, no se ha encontrado ninguna respuesta y se va al paso 7.

6.3 Si es mayor que uno, entonces, existe una contradicción o ambigüedad y se procede al cálculo del VE del MI con cada uno de los marcos almacenados en la lista de marcos CANDIDATOS. Aquel marco que obtenga el VE máximo es el seleccionado. Posteriormente, se va al paso 7.

Paso 7: En este paso, el algoritmo finaliza.

5.3.2.4 Donación de Propiedades

5.3.2.4.1. Tipos de Inferencias

En esta sección se propone una nueva técnica, llamada *Donación*, que utiliza las relaciones "ad hoc" para realizar inferencias basadas en la cesión de propiedades entre marcos instanciados conectados por dichas relaciones.

La Donación, hace a los sistemas más inteligentes al capturar el conocimiento de sentido común asociado a las relaciones "ad hoc" definidas en la BC. Definida una relación "ad hoc" con los elementos descritos en el apartado 5.2.2.1.2, las inferencias asociadas a cualquier relación de este tipo se clasifican en:

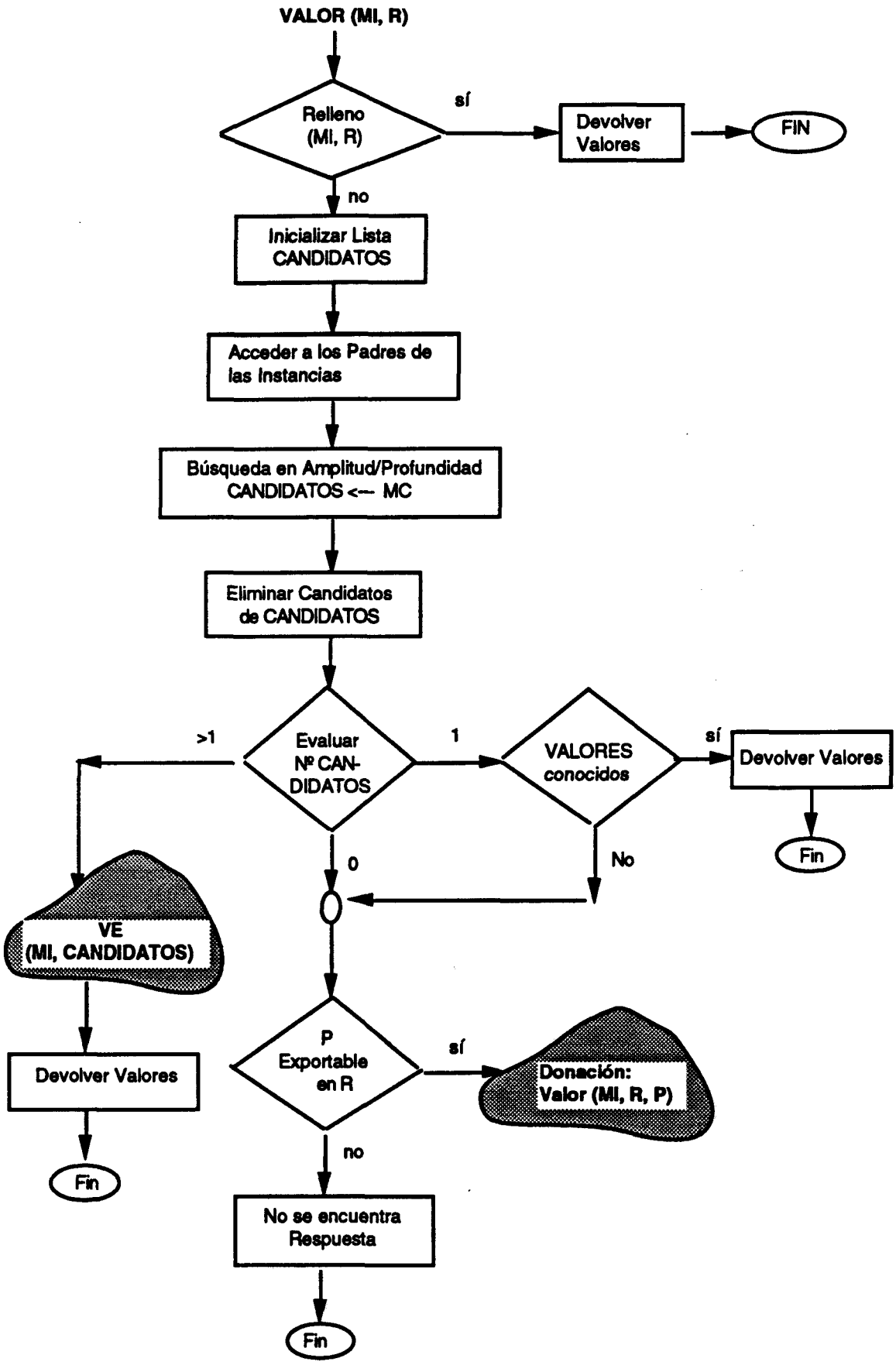


Figura 5.42. Algoritmo de Herencia

A) Las que se realizan al relacionar o conectar marcos instanciados mediante relaciones "ad hoc", y

B) Las que ocurren al ejecutar el sistema.

A) Inferencias al Construir la BC.

Definida una relación R entre uno o varios marcos clase, las inferencias que se realizan al conectar una relación de este tipo entre un marco origen y un marco destino, ambos instanciados, son:

A.1) Compatibilidad de tipos.

Se comprueba que el marco instanciado origen de la relación es una instancia del marco que se ha definido como origen de la relación, y, que el marco instanciado destino de la relación es una instancia del marco destino al cual llega la relación.

A.2) Relación Inversa o Recíproca.

Por cada relación "ad hoc" que conecte dos marcos instanciados se debe crear la relación inversa o recíproca entre dichos marcos instanciados. Pueden darse dos situaciones:

* Si la relación R se ha definido simétrica, entonces se creará la relación recíproca con el mismo nombre.

* Si no lo es, se creará con el nombre de la relación inversa dado por el IC al definir la relación "ad hoc" en la BC.

B) Inferencias al Ejecutar el SBC

El alcance de las inferencias que realizan las relaciones "ad hoc" al ejecutarse en un SBC está supeditado a la semántica de las propiedades exportables definidas en la relación y a los requisitos que deben cumplir los valores de dichas propiedades para ser cedidas a otros marcos instanciados. Estos requisitos, como ya se vió en la definición de las relaciones "ad hoc", se expresan en la faceta *Precondición* de la

propiedad que se cede mediante un conjunto de condiciones que deben cumplir las propiedades de los marcos instanciados que la relación une.

Supóngase la jerarquía de la figura 5.43. Si en un momento dado se desea conocer cual es el valor que toma la propiedad P en la relación R en el marco instanciado MI_1, el sistema realizará cesión de propiedades basada en Donación y, opcionalmente, Herencia.

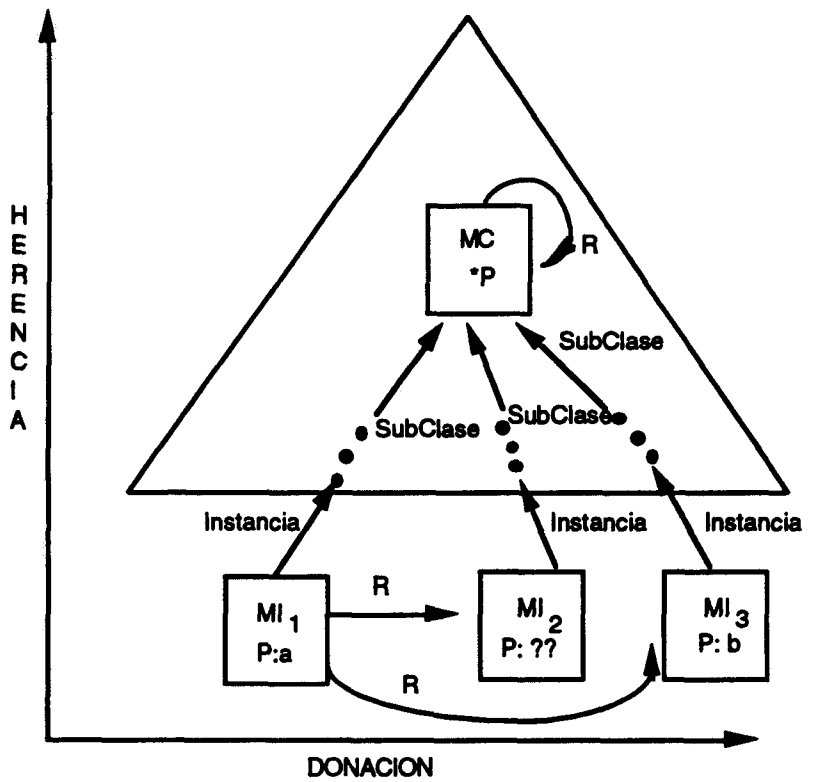


Figura 5.43. Dos dimensiones de la Cesión de Propiedades.

Estas inferencias se combinan dando lugar a una herencia híbrida en dos dimensiones:

1. *Dimensión Horizontal (Donación).* Si la propiedad P se ha rellenado en el marco instanciado con unos valores locales al marco, y si éstos valores cumplen las restricciones impuestas en la definición de la relación, entonces, los valores de la propiedad en el marco instanciado son valores exportables en la relación R, y, por consiguiente, son donados a otros marcos instanciados.

En el ejemplo de la figura 5.43, el valor *b* de la propiedad P del marco instanciado MI_3 se dona (Horizontal) al marco instanciado MI_1 siempre que *b* cumpla las restricciones impuestas en la definición de la propiedad exportable en R.

2. *Dimensión Vertical (Herencia).* Si la propiedad P no se ha rellenado en el marco instanciado, entonces, se aplica herencia de propiedades con el fin de asignar un valor local a la propiedad del marco instanciado. Si el usuario o el SBM utilizando valores por omisión le asigna un valor, entonces, se realizará la donación de propiedades según lo descrito en la situación 1 siempre que se cumplan las restricciones impuestas en la definición de la relación. Si la propiedad, tras aplicar herencia, no toma ningún valor local en el marco instanciado, no puede plantearse la donación de valores.

   En el ejemplo anterior, en el marco instanciado MI_2 se aplica herencia (Vertical) de propiedades para asignar un valor local a la propiedad P. Una vez asignado un valor, y si estos valores cumplen las restricciones, se donan (Horizontal) al marco instanciado MI_1.

La búsqueda de los valores que toma una propiedad exportable P en una relación R para un marco instanciado MI, debe realizarse en todo el grafo conexo que tiene como origen el marco instanciado para el cual se realiza la consulta. El procedimiento que recorre el grafo de marcos instanciados utiliza las siguientes tres listas:

1. De VALORES:

   Almacena el conjunto de valores con los que se ha rellenado la propiedad P en todas las instancias conectadas con la instancia actual, valores que satisfacen las restricciones impuestas en la definición de la propiedad exportable de la relación R.

2. De marcos CANDIDATOS:

   Conjunto de marcos instanciados que están conectados con el marco actual y que aún no se han analizado.

3. De marcos ANALIZADOS:

   Conjunto de marcos instanciados en los que ya se ha buscado la propiedad P.

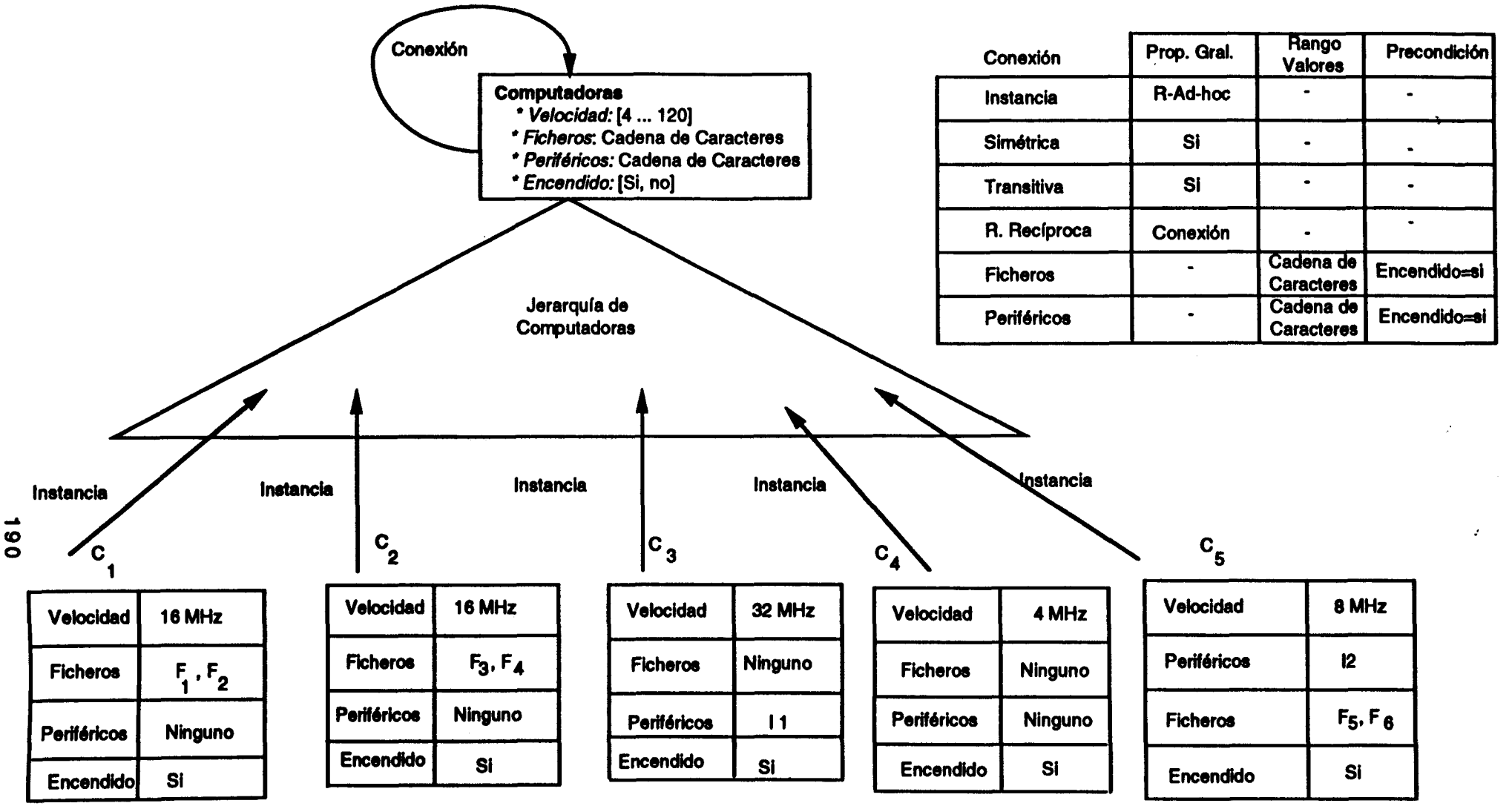
Considérese de nuevo la relación *Conexión*. Supóngase que el IC define, al construir la BC, una relación "ad hoc" llamada *Conexión* en el marco clase *Computadora* que tiene, como propiedades exportables, las propiedades *Ficheros* y *Periféricos*. La figura 5.44.(a), muestra la jerarquía de computadoras y la definición de la relación *Conexión* como una relación "ad hoc". En dicha figura, el IC ha rellenado algunas propiedades miembro definidas en los marcos clase con los que las instancias están conectadas, pero aún no ha conectado unos marcos instanciados con otros. Supóngase que en este momento decide empezar a conectar los marcos instanciados en el orden dado por su numeración.

Al rellenar la ranura *Conexión* del marco instanciado *Computadora1* con los valores: *Computadora2* , *Computadora3* y *Computadora4*, se comprueba, en primer lugar, la compatibilidad de tipos y se crean las tres relaciones recíprocas asociadas entre las computadoras, introduciendo en las ranuras *Conexión* de las instancias: *Computadora2*, *Computadora3* y *Computadora4* el valor *Computadora1*, como, gráficamente, se muestra en la figura 5.44.(b).

El siguiente paso que realiza el IC es conectar el marco instanciado *Computadora2* con el marco instanciado *Computadora3*. Automáticamente, se comprueba la compatibilidad de tipos y se crea la relación recíproca que conecta el marco instanciado *Computadora3* con el marco instanciado *Computadora2*. La figura 5.44.(c) muestra este conocimiento. El proceso continúa para todas las conexiones de ordenadores definidas en la red de la figura 5.26. El resultado es la jerarquía de marcos en la figura 5.44.(d).

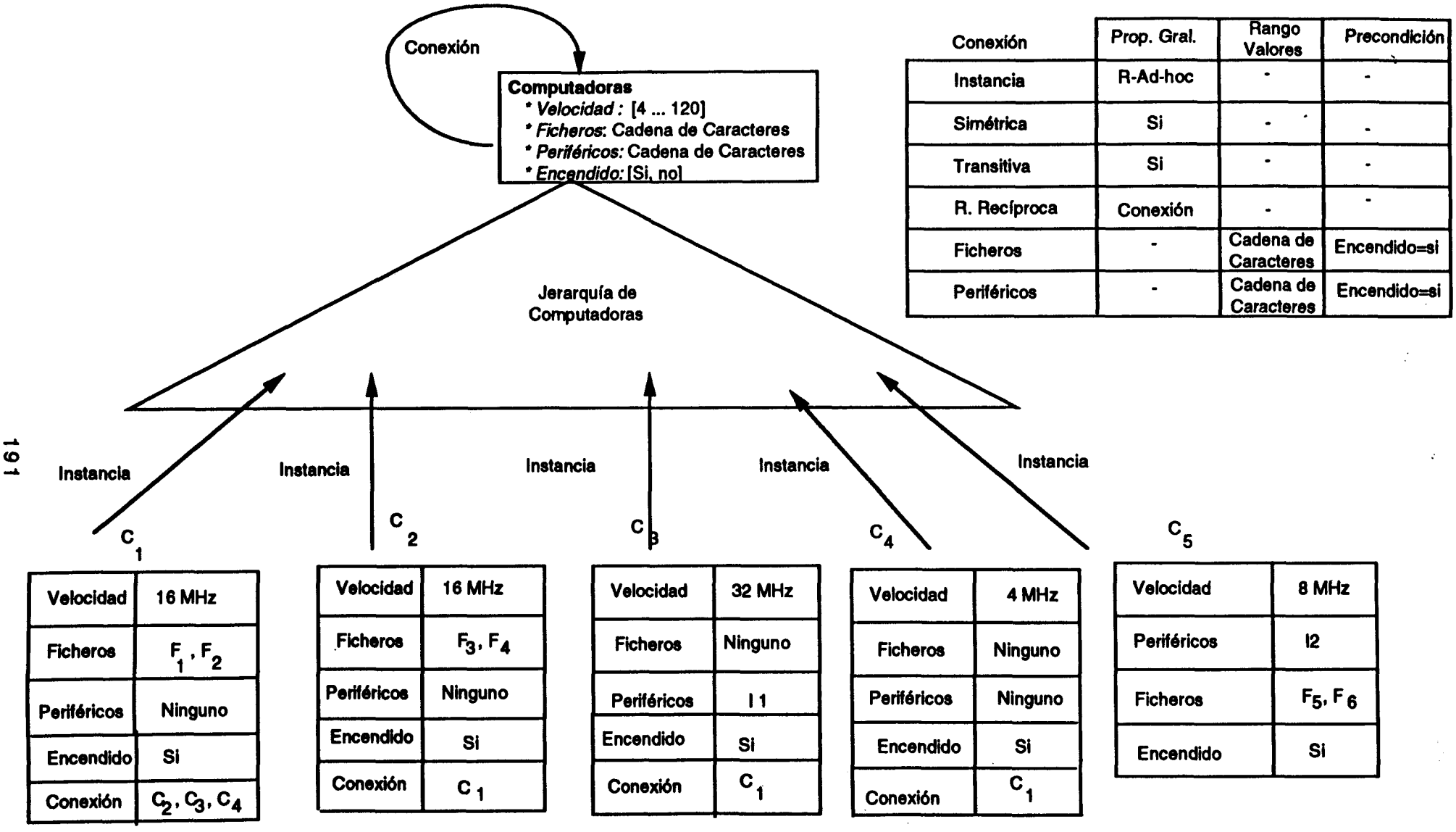
De esta forma, si una vez definida la relación *Conexión* como una relación simétrica y transitiva, y, habiendo definido las propiedades *Ficheros* y *Periféricos* como propiedades exportables no sujetos a ningunas restricciones junto con la ranura *Conexión*, se realizan las anteriores consultas 12 y 13, las respuestas que se obtienen son las mismas.

Como conclusión de este apartado se puede decir que las inferencias que se realizan sobre relaciones "ad hoc" incorporan al sistema conocimiento de sentido común que el IC no introducía antes al formalizar BC. Disminuyendo así la pérdida de conocimiento que se produce al formalizar BC basadas en Marcos.



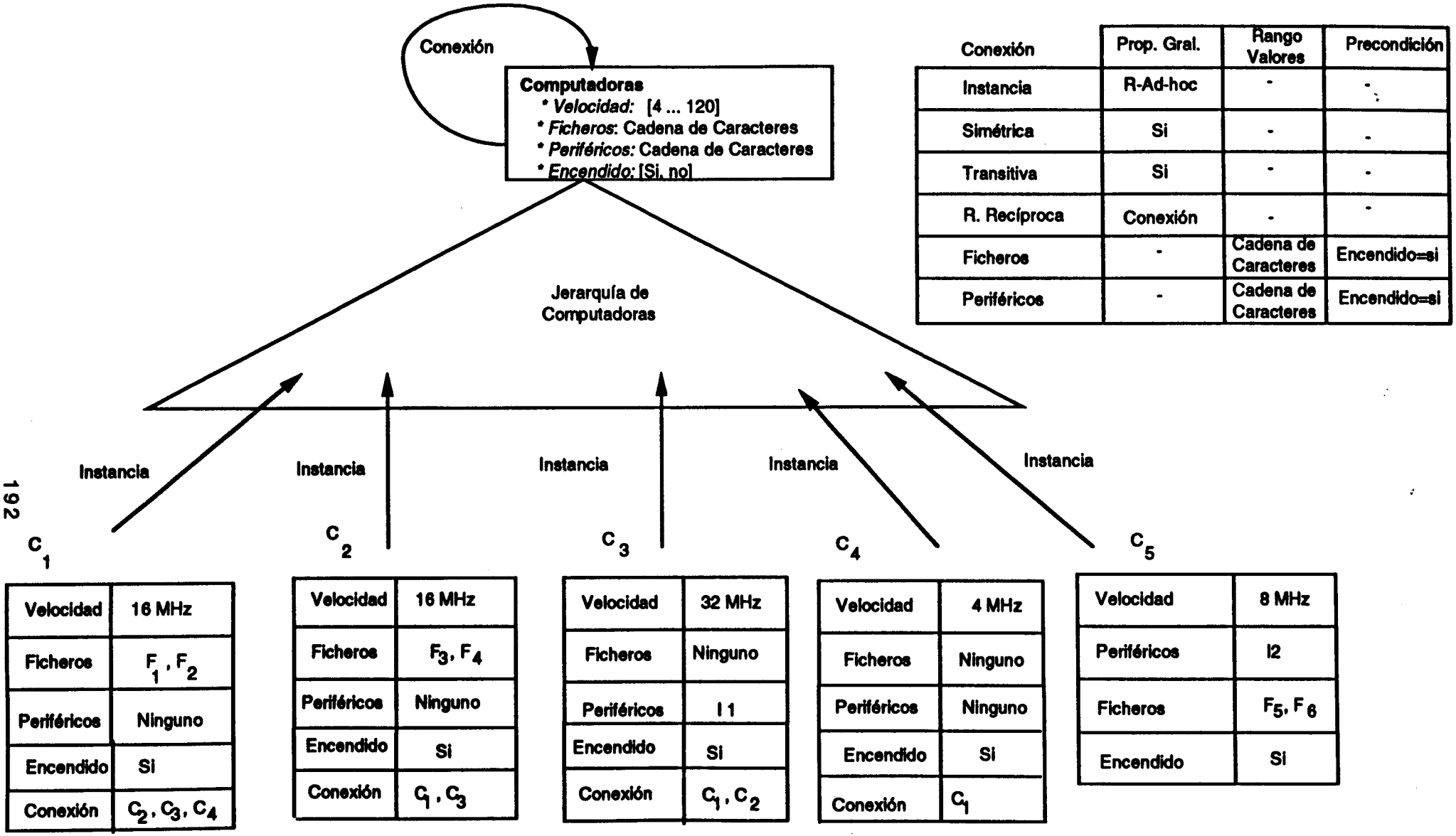
Conexión	Prop. Gral.	Rango Valores	Precondición
Instancia	R-Ad-hoc	-	-
Simétrica	Si	-	-
Transitiva	Si	-	-
R. Recíproca	Conexión	-	-
Ficheros	-	Cadena de Caracteres	Encendido=si
Periféricos	-	Cadena de Caracteres	Encendido=si

Figura 5.44.(a). SBM para una red de computadoras.



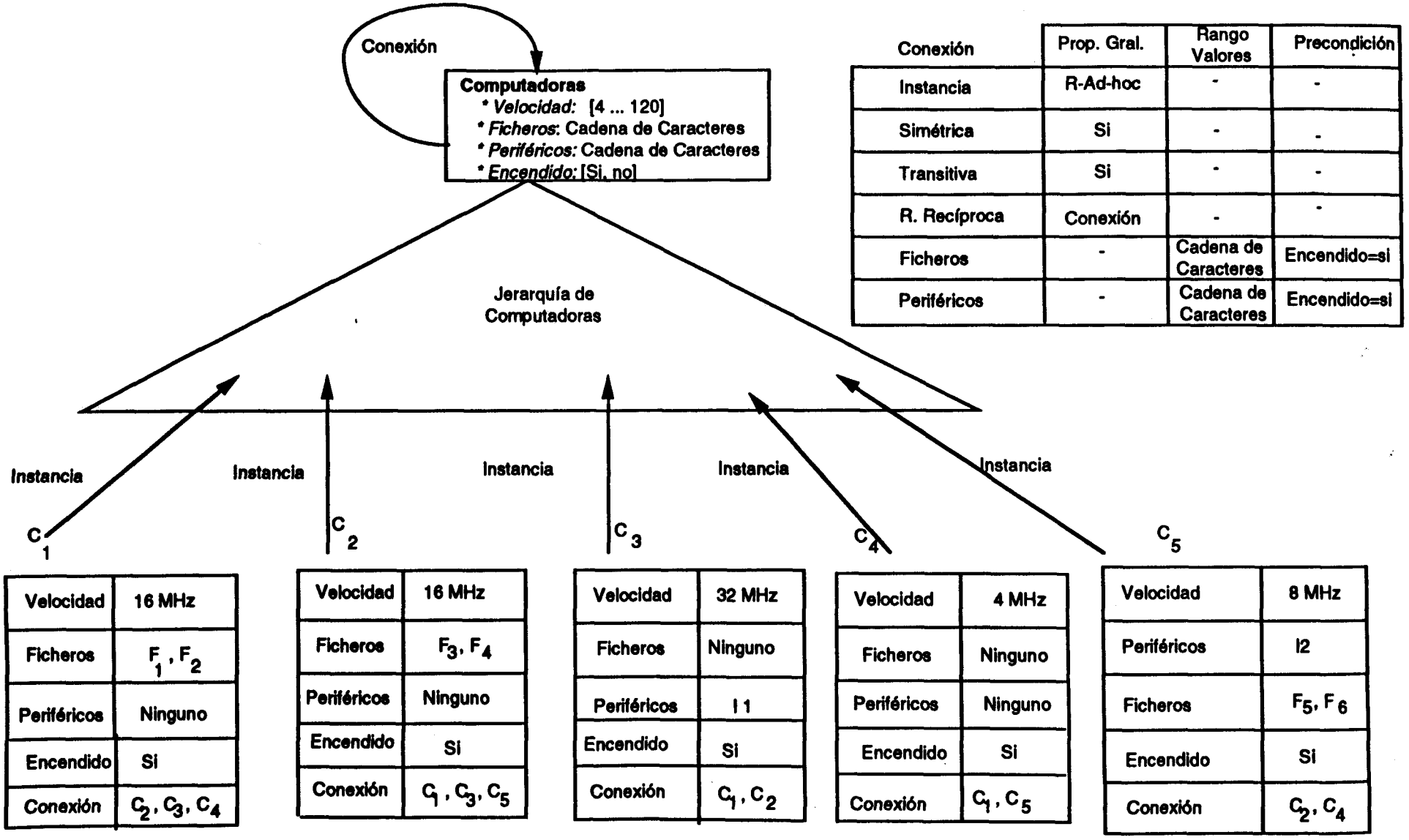
Conexión	Prop. Gral.	Rango Valores	Precondición
Instancia	R-Ad-hoc	-	-
Simétrica	Si	-	-
Transitiva	Si	-	-
R. Recíproca	Conexión	-	-
Ficheros	-	Cadena de Caracteres	Encendido=si
Periféricos	-	Cadena de Caracteres	Encendido=si

Figura 5.44.(b). SBM para una red de computadoras con tres conexiones.



Conexión	Prop. Gral.	Rango Valores	Precondición
Instancia	R-Ad-hoc	-	-
Simétrica	Si	-	-
Transitiva	Si	-	-
R. Recíproca	Conexión	-	-
Ficheros	-	Cadena de Caracteres	Encendido=si
Periféricos	-	Cadena de Caracteres	Encendido=si

Figura 5.44.(c). SBM para una red de computadoras.



Conexión	Prop. Gral.	Rango Valores	Precondición
Instancia	R-Ad-hoc	-	-
Simétrica	Si	-	-
Transitiva	Si	-	-
R. Recíproca	Conexión	-	-
Ficheros	-	Cadena de Caracteres	Encendido=si
Periféricos	-	Cadena de Caracteres	Encendido=si

Figura 5.44.(d). SBM para la red completa.

5.3.2.4.2 *Algoritmo de Donación*

El algoritmo que aquí se propone para realizar donaciones de propiedades entre marcos instanciados, asociado al organigrama de la figura 5.45, es el siguiente:

Paso 1: Inicializar las listas de VALORES, de marcos CANDIDATOS y de marcos ANALIZADOS con vacía.

Paso 2: Si la propiedad P se ha rellenado en el marco instanciado MI, entonces, se almacenan los valores en la lista de VALORES. Si la propiedad no tiene asignada ningún valor, se llama al algoritmo de herencia. Posteriormente, se almacenan en la lista de marcos CANDIDATOS los marcos instanciados con los que se ha rellenado la relación R en el marco instanciado actual.

Paso 3: Si el uso de la relación R está permitido en el marco instanciado se va al Paso 4. En caso contrario, se va al Paso 8.

Paso 4: Si en la relación R se ha definido la propiedad P como exportable, entonces, se va al Paso 5. En caso contrario, se va al Paso 8.

Paso 5: Para cada marco instanciado de la lista de marcos CANDIDATOS, se realizan los pasos 6 y 7. Si la lista de marcos CANDIDATOS está vacía, se va al Paso 8.

Paso 6: Se almacena en la lista de VALORES, los valores con los que se ha rellenado la propiedad P en el marco instanciado, siempre que se cumplan las restricciones definidas por el IC en la faceta *Precondición* de la propiedad exportable en la relación R. Pueden darse dos casos:

6.1. Si la propiedad P se ha rellenado en el marco instanciado, entonces, se almacenan los valores asociados a la propiedad en una lista de VALORES siempre que no estén incluidos en dicha lista y se cumplan las condiciones impuestas por el IC. Posteriormente, se va al Paso 7.

6.2 Si la propiedad P no se ha rellenado en el marco instanciado, entonces, aplicando la herencia de propiedades se intenta dar valores a la propiedad P. Si la propiedad se ha rellenado, y se cumplen las condiciones impuestas por el IC, se incluyen los valores en la lista de VALORES simpre que no estén ya en ella. En cualquier caso, se va al paso 7.

Paso 7: Se almacena en la lista de marcos CANDIDATOS los marcos instanciados con los que se ha rellenado la relación R en el marco instanciado actual que no se encuentren en la lista de ANALIZADOS.

7.1 Si la ranura que representa la Relación R se ha rellenado en el marco instanciado, se almacenan los marcos candidatos que no están analizados en la lista de marcos CANDIDATOS y el marco actual en la lista de marcos ANALIZADOS. Posteriormente, se va al paso 5.

7.2 Si la ranura que representa la Relación R no se ha rellenado en el marco instanciado, entonces, aplicando la herencia de propiedades, se ejecuta el procedimiento que intenta dar valores a la relación. Si ésta se rellena, se almacenan los marcos instanciados no analizados en la lista de marcos CANDIDATOS y el marco actual en la lista de marcos ANALIZADOS y se va al paso 5. Si no se rellena la ranura, también se va al paso 5.

Paso 8: Devolver los valores almacenados en la lista de VALORES y se finaliza.

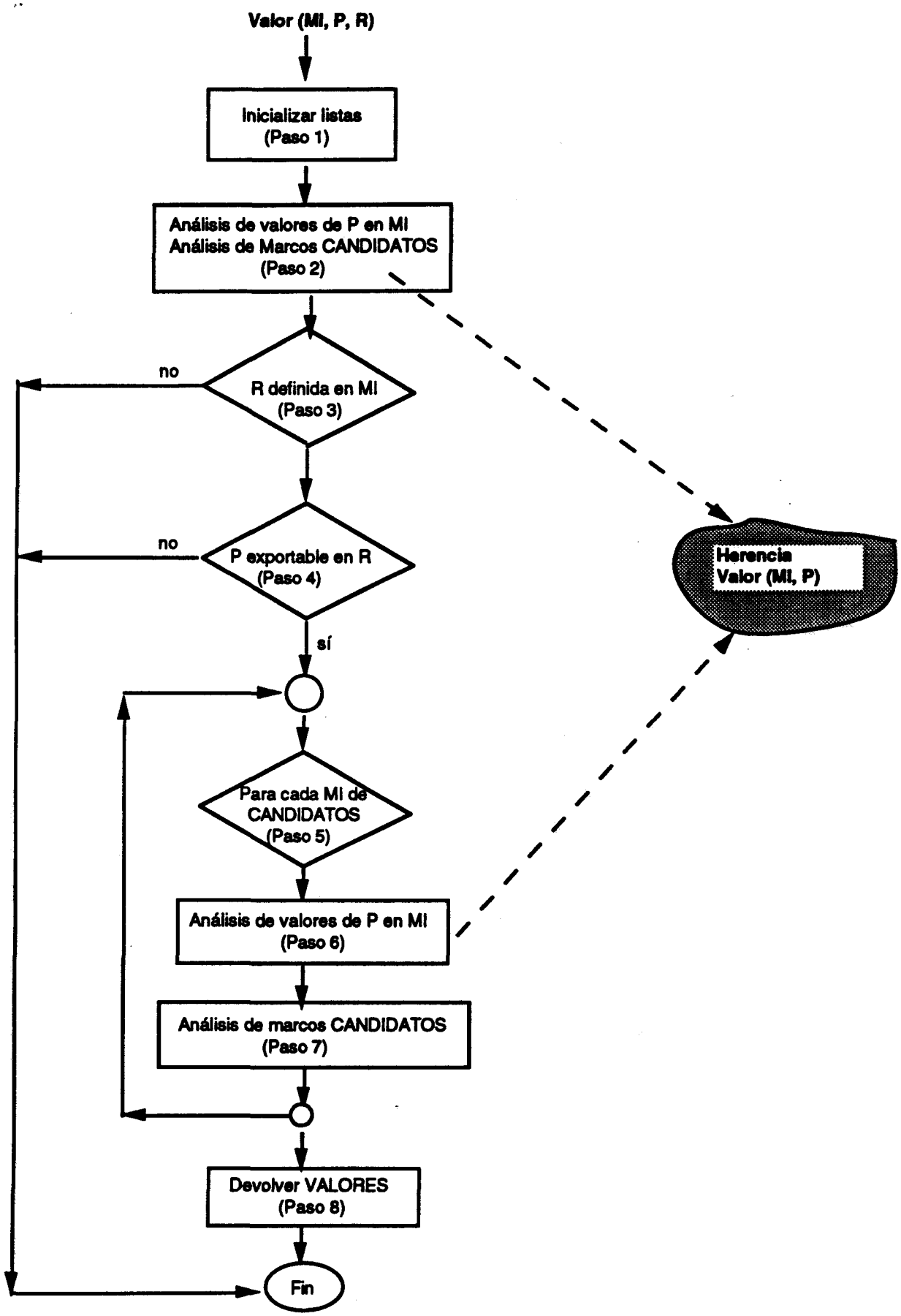


Figura 5.45. Organigrama de Donación

5.4. DISEÑO DE UN ENTORNO ORIENTADO A MARCOS

Como aplicación práctica del Modelo de Diseño desarrollado en el apartado 5.2, se ha construido un entorno que permite formalizar BC utilizando como técnica de representación del conocimiento el formalismo de Marcos. El entorno es un SBC basado en Marcos programado con orientación a objetos. Se trata de una cierta anidación. Es decir, son marcos que permiten formalizar cualquier SBC en marcos.

La idea de este entorno es la siguiente: El IC, utilizando la representación tabular descrita en la tabla 5.1, construye taxonomías o jerarquías de marcos en un dominio. Con el fín de garantizar las restricciones sintácticas y semáticas descritas en el apartado 5.2, se han construido varias jerarquías de conceptos en las cuales se han formalizado cada uno de los conceptos que forman el paradigma. A estas jerarquías, genéricamente, se las ha llamado Jerarquía de Conceptos de Marcos y en ella se han representado los marcos, sus propiedades y relaciones como marcos. Junto con la jerarquía de conceptos de marcos, otras jerarquías también basadas en marcos forman parte del entorno: las jerarquías de programas, sentencias y operaciones. Estas jerarquías de marcos permiten expresar cualquier programa, sentencia u operación que aparezca en la definición de una propiedad como marcos instanciados. La única jerarquía del entorno no representada en marcos es la jerarquía de Tipos. Esta jerarquía es una jerarquía de objetos que representa los tipos básicos utilizados tanto en marcos como en los programas. Se consigue así representar declarativamente el conocimiento procedimental del paradigma. En la figura 5.46 se representan las tres jerarquías: la jerarquía de la aplicación, la jerarquía de conceptos de marcos y la jerarquía de programas, y cómo se relacionan.

5.4.1. ENTORNO ORIENTADO A MARCOS

Los pasos que se han seguido en el diseño y posterior implementación del entorno, han sido:

1. Se han representado todos los conceptos que aparecen descritos en el apartado 5.2 en una jerarquía de marcos. A esta jerarquía se la llamará, de ahora en adelante, **Jerarquía de Conceptos de Marco (JCM).**

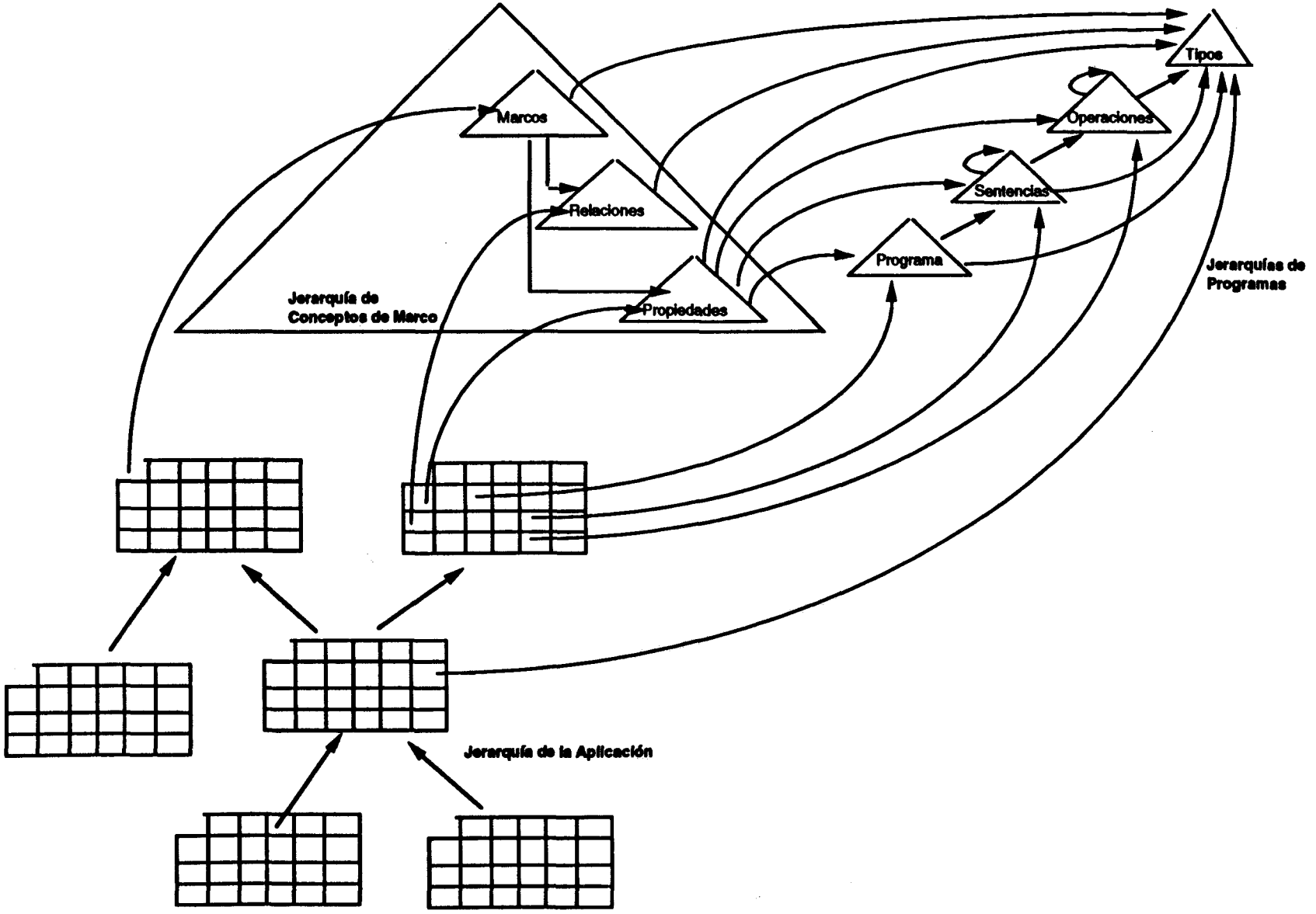


Figura 5.46 Relaciones entre Jerarquías

En la JCM se han representado como marcos los conceptos: Marco Clase, Marco Instanciado, Ranuras, Propiedades y Relaciones. De ahora en adelante, cuando se haga referencia a los marcos que representan estos conceptos, dichos marcos se escribirán en *cursiva*. Por ejemplo, los marcos clases que se han definido en una aplicación concreta son instancias del marco *Marco Clase* en la JCM, sus marcos instanciados instancias del marco *Marco Instanciado*, las propiedades de clase instancias del marco *Propiedad de Clase*. En general, cualquier elemento que se defina en una BC de una aplicación será instancia de un marco definido en la JCM. La JCM permite que el IC construya bases de conocimientos de acuerdo a la base teórica descrita en el apartado 5.2, y a la vez, la jerarquía JCM impide la definición y uso indebido de ellos. Se asegura así la coherencia lógica y sintáctica de los elementos que forman la BC de una aplicación concreta.

2. Se han desarrollado procedimientos que realizan **Inferencias** sobre cualquier BC construida sobre la JCM. Concretamente, se han implementado en C++, los algoritmos descritos en el apartado 5.3 al tratar la equiparación, la herencia y la donación de propiedades.

   Con estos procedimientos el IC puede ir comprobando en su BC, a medida que profundiza en la formalización de su sistema, si las inferencias que el formalismo tiene asociadas razonan con el conocimiento de forma similar a como el experto razona en el dominio de la aplicación. Si es así, se tiene garantizado, en la etapa de formalización, que los marcos son un buen formalismo para representar el conocimiento de la aplicación que se está desarrollando. En caso contrario, se sabe que el formalismo de marcos no representa de manera natural el conocimiento del dominio. En ambos casos, no ha sido necesario implementar el prototipo.

3. Se ha construido un compilador que genera, para cualquier programa, ecuación o expresión lógica, un conjunto de marcos instanciados que se enlazan en ejecución y que producen el mismo resultado que si el programa, la ecuación o la expresión, fuera codificada y ejecutada en un lenguaje de alto nivel. Con ello se consigue traducir el conocimiento procedimental almacenado en el marco al formalismo de marcos y tener todo el conocimiento declarativo y procedimental del formalismo almacenado declarativamente en la Base de Conocimientos.

En el anexo II se muestra cómo se ha diseñado este compilador que genera un conjunto de marcos instanciados para representar, utilizando el formalismo de marcos, cualquier programa escrito en un lenguaje de alto nivel. En dicho anexo, se detallan varias jerarquías que representan los tipos básicos más utilizados en las declaraciones de variables, las operaciones aritméticas y lógicas, las sentencias de asignación, iterativas, condicionales y de entrada/salida, y los programas. Cada variable, operación o sentencia, que aparezca en un programa será instancia de algún marco clase de alguna de estas jerarquías.

También se detalla en el anexo II, la traducción de varios programas a un conjunto de marcos instanciados y se explican su funcionamiento en ejecución. A grosso modo, la ejecución del programa se realiza mediante una cadena de métodos que se envían los marcos instanciados que representan el orden lógico en el que se encuentran las sentencias escritas en el programa. Los marcos instanciados, se ejecutan en ese momento con el valor que toman las variables en el programa.

4. Se ha construido un interfaz de usuario que permite al IC de la aplicación formalizar Bases de Conocimientos utilizando la estructura tabular propuesta en el apartado 5.2.

Se consigue con este entorno que el IC formalice bases de conocimientos basadas en Marcos con la misma expresividad que aquél utilizaba cuando formalizaba manualmente. Sin embargo, las diferencias y las grandes ventajas que este entorno proporciona al IC son:

* Todo el conocimiento declarativo formalizado es coherente sintáctica y lógicamente. Cualquier sistema así construido cumplirá las restricciones sintácticas y semánticas impuestas en la teoría, del apartado 5.2, al impedir la definición de elementos no permitidos en ella.

* El conocimiento procedimental puede ser expresado en un lenguaje de alto nivel. El IC puede evaluar si el procedimiento trabaja correctamente con el conocimiento declarativo de la BC y si realiza las inferencias adecuadas.

* El conocimiento formalizado en este entorno se convierte en un prototipo de la aplicación al poder razonar las técnicas de inferencias descritas en el apartado 5.3 sobre la BC de la aplicación. De esta forma, el IC comprueba si el formalismo de representación y sus técnicas de inferencia asociadas modelan fielmente la realidad o no.

* El entorno, utilizando el SBM sobre el cual está construido, y razonando sobre el conocimiento almacenado en la BC de la aplicación, incluirá nuevo conocimiento en la BC y permitirá al IC validar el conocimiento del dominio en la fase de formalización, en vez de tener que esperar a que la BC esté totalmente implementada.

5.4.2. CONOCIMIENTO DECLARATIVO: JERARQUIA DE CONCEPTOS DE MARCO

En este apartado se describen todos los conceptos del apartado 5.2 utilizando el Formalismo de Marcos. El resultado, es una BC formalizada en marcos en la JCM. Esta jerarquía contiene todo el conocimiento declarativo del formalismo expresado declarativamente y está formada por las jerarquías de marcos, relaciones y propiedades, como gráficamente se ha mostrado en la figura 5.46. El significado de estas jerarquías es el siguiente:

1. Jerarquía de Marcos

   Conjunto de marcos clase que define los tipos de Marcos permitidos en el formalismo. Cada tipo de Marco viene determinado por el conjunto de relaciones que le conectan con otros marcos y por un conjunto de propiedades que definen el concepto que el marco representa. Por tanto, relaciones y propiedades son los atributos que definen cualquier Marco.

2. Jerarquía de Relaciones

   Jerarquía de marcos clase que define el conjunto de relaciones permitidas en el paradigma de marcos.

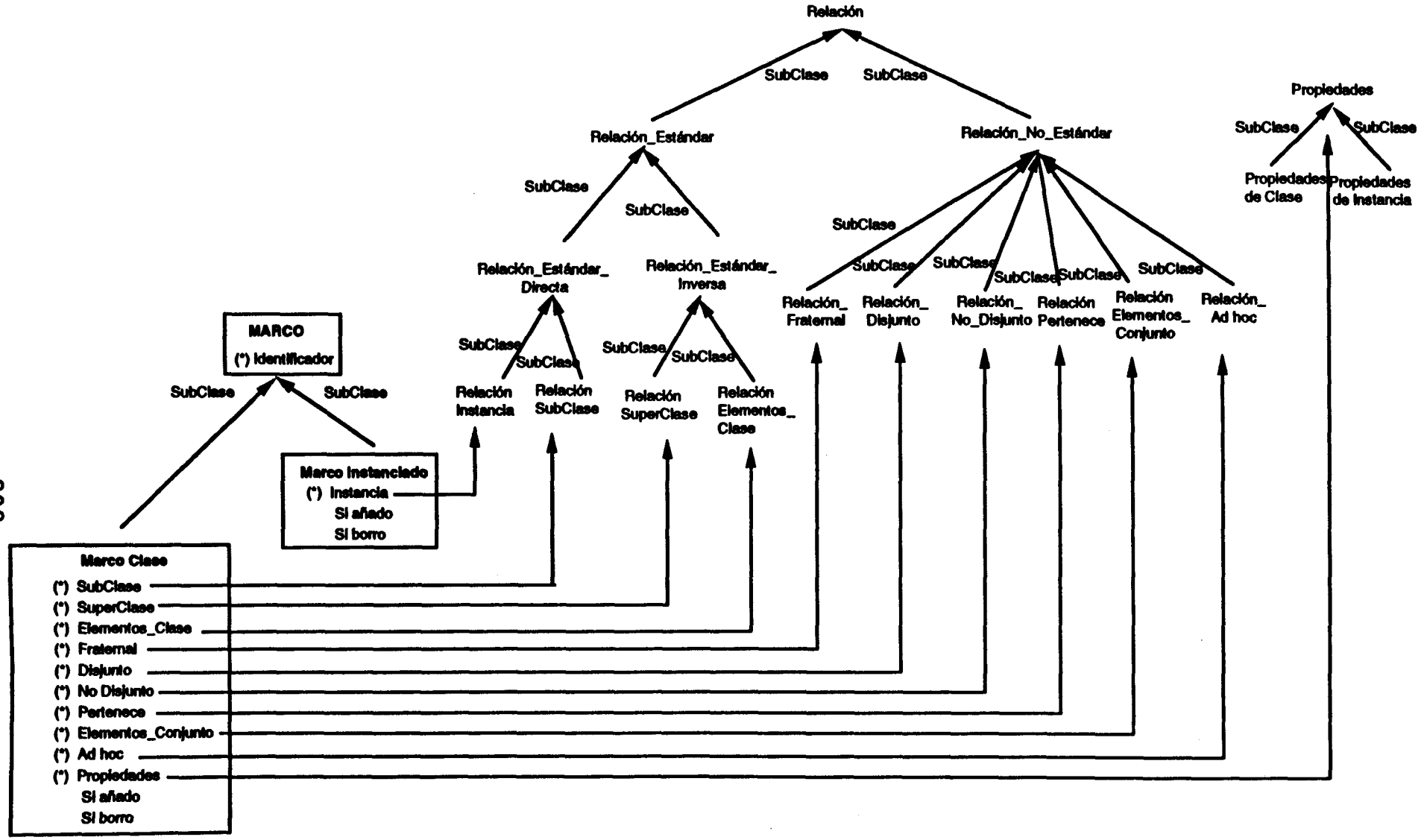


Figura 5.47. Jerarquía de Conceptos de Marcos

3. Jerarquía de Propiedades

Define los tipos de propiedades presente en cualquier SBM. Los atributos que definen las propiedades son las facetas. Por tanto, cada tipo de propiedad tendrá unas facetas asociadas en esta jerarquía.

Estas tres jerarquías están conectadas como gráficamente se muestra en la figura 5.47. Obsérvese que las propiedades (relaciones y propiedades del formalismo) definidas en los marcos *Marco Clase* y *Marco Instanciado* se definen como punteros a marcos clase de la jerarquía de relaciones y de propiedades.

La ventaja de este entorno, es que cualquier marco, propiedad o relación que se defina en una BC de una aplicación, deberá cumplir las restricciones impuestas en el formalismo y expresadas declarativamente en la JCM. Por ello:

* Los marcos, propiedades o relaciones que se empleen en el desarrollo de la aplicación serán instancias de un marco clase de la JCM.

* Por ser instancia de un marco clase, heredará el conjunto de propiedades definidas en él.

5.4.2.1. Jerarquía de Marcos

La JCM es un conjunto de marcos clase que describen, entre otros, los conceptos de *Marco Clase* y de *Marco Instanciado*. Cualquier marco clase o instancia que se cree en una BC será instancia de alguno de estos marcos y, por consiguiente, heredará las propiedades en ellos definidas.

Por ejemplo, el marco clase *Persona* es una instancia del marco *Marco Clase* y va a heredar todas las propiedades y métodos definidos en él. *Juan*, que es una instancia de *Persona*, e instancia del marco *Marco Instanciado*, heredará las propiedades y métodos definidos en el marco *Persona* y en el marco *Marco Instanciado*. Gráficamente, se representan estos conceptos en la figura 5.48.

Como ya se vió en el apartado 5.2, un marco se relaciona con otros marcos utilizando un conjunto de relaciones estándares y no estándares. Con este fin, se va a describir el conjunto de relaciones permitidas asociadas a cada tipo de marco. Las

figuras 5.49, y 5.50 muestran el conjunto de relaciones potenciales que pueden salir de cualquier marco clase o de cualquier marco instanciado.

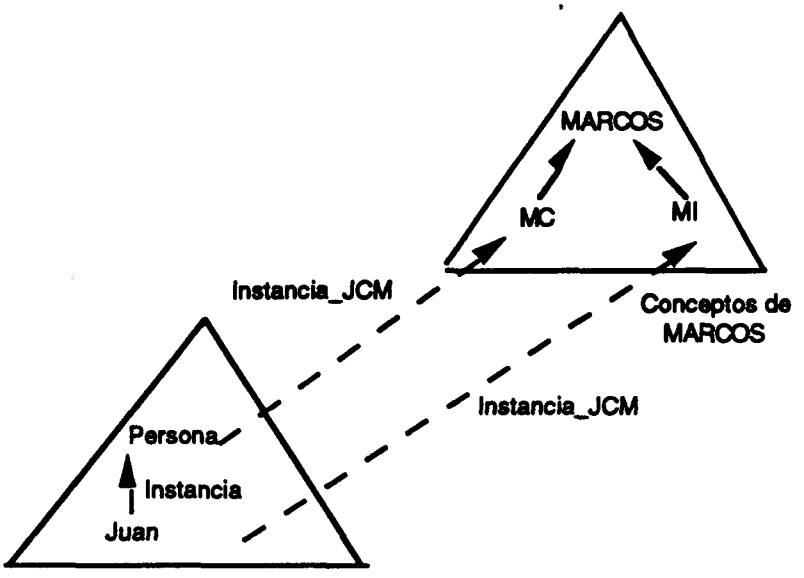


Figura 5.48. *Juan* instancia de *Persona* e instancia del concepto *Marco Instanciado*

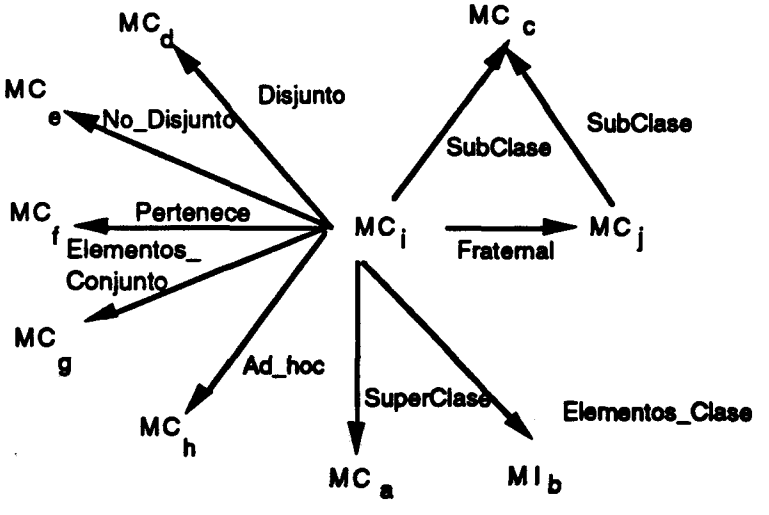


Figura 5.49. Relaciones que salen de los marcos clases.

Figura 5.50. Relación que sale de los marcos instanciados.

El estudio detallado que se ha realizado de los marcos lleva a establecer un conjunto de características comunes a todos ellos.

a) Cualquier marco tiene un identificador,

b) Los marcos se relacionan con otros marcos,

c) En los marcos se definen propiedades, y

d) Las operaciones básicas que se realizan sobre los marcos son: *Crear_Marco* y *Borrar_Marco*.

La *Jerarquía de Marcos* que representa el concepto de Marco es la de la figura 5.51.

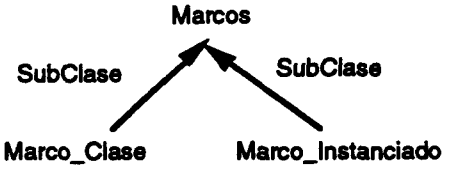


Figura 5.51. Jerarquía de Marcos.

Asociado a cada marco existe un conjunto de características que lo definen como: los marcos con los que está relacionado y las propiedades definidas en él. Por tanto, relaciones y propiedades son las propiedades de instancia definidas en el marco *Marco*. Gáficamente, utilizando la notación de Marcos, un marco se corresponde con las plantillas que se muestran en la figura 5.52. Las tablas 1, 2 y 3 del anexo III describen en profundidad cada uno de estos marcos de la Jerarquía de Marcos.

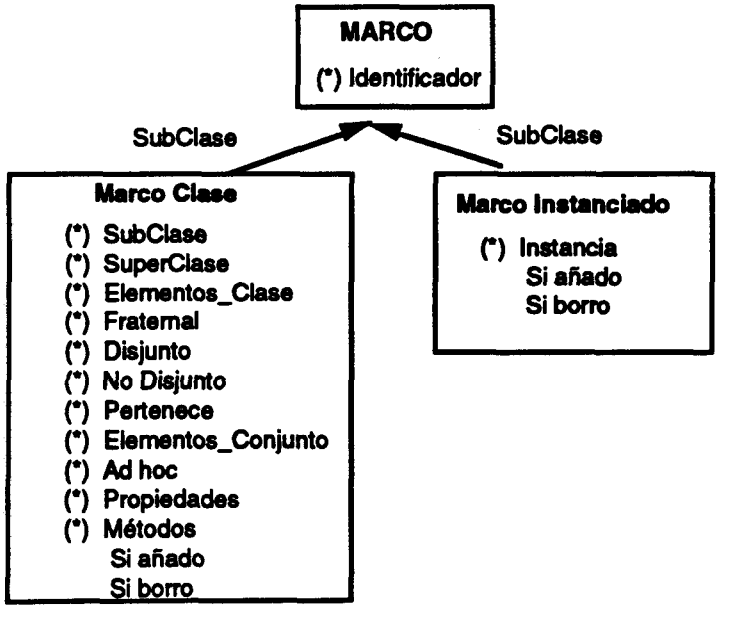


Figura 5.52. Propiedades de los marcos de la Jerarquía de Marcos

Por ejemplo, en la tabla 3 que representa el concepto *Marco Clase*, las propiedades *SubClase* y *Fraternal* se han definido utilizando los marcos *Relación SubClase* y *Relación Fraternal*, y los valores con los que se rellenarán estas propiedades en los marcos instanciados del marco *Marco Clase* serán los punteros a instancias de dichos marcos.

Las operaciones básica *Crear_Marco_Clase*, *Borrar_Marco_Clase*, *Crear_Marco_Instanciado* y *Borrar_Marco_Instanciado* asociadas a los métodos *Si añado* y *Si borro* de los marcos *Marco Clase* y *Marco Instanciado* se ejecutan cuando se solicita la creación o el borrado de un marco clase o de un marco instanciado en una BC concreta. La definición de estos procedimientos se ha realizado en el apartado segundo del anexo III, siguiendo el esquema del conocimiento procedimental (precondición, acción, postcondición) descrito en la sección 5.2.2.2.2. El esquema de la definición de los métodos que crean y borran marcos es el mostrado en la figura 5.53. Obsérvese que estos procedimientos se limitan a crear o borrar un marco instanciado, o un marco clase, siempre que se cumplan las precondiciones, impidiendo la creación de los marcos si éstos ya existen o el borrado en caso contrario.

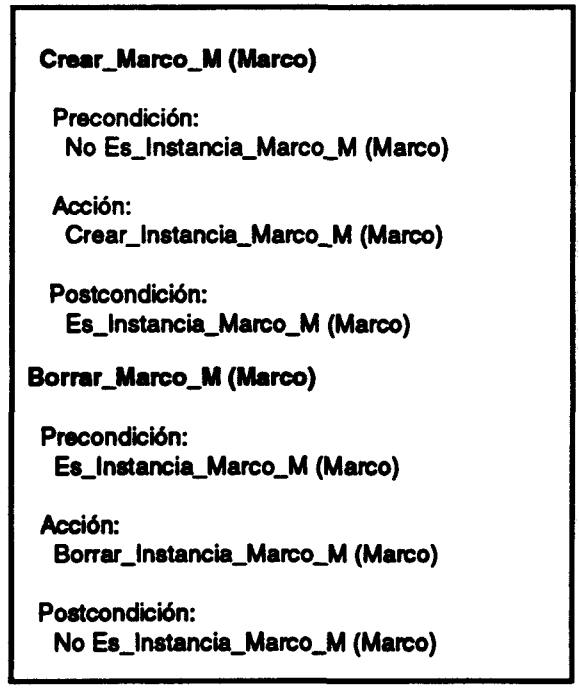


Figura 5.53 Esquema de la definición de los métodos de los Marcos

5.4.2.2. Ranuras

5.4.2.2.1. Relaciones

El estudio detallado que se ha hecho de las relaciones lleva a establecer un conjunto de propiedades o características comunes a todas ellas:

a) Toda relación tiene un nombre o identificador

b) En toda relación interviene un origen y un destino

c) Toda relación tiene asociada una relación inversa o una relación recíproca

d) Las operaciones básicas que se realizan con las relaciones son: *Crear_Relación* y *Borrar_Relación*.

Por ejemplo, si se tiene un marco clase *Persona* y un marco clase *Mujer* en una BC y se define la relación *Mujer SubClase Persona*, se estaría solicitando la ejecución de la operación: *Crear_Relación_SubClase (Mujer, Persona)*. Pero, si en vez de haber definido una relación SubClase se hubiera definido una relación Instancia entre *Mujer* y *Persona*, el procedimiento que crea la relación debería indicar que la relación no cumple los requisitos sintácticos del formalismo y, por consiguiente, la relación no está permitida.

Dado que para cada relación se ha definido una relación inversa o recíproca y un conjunto de restricciones, la definición de las operaciones *Crear* y *Borrar* en cada relación se realiza, de manera natural, utilizando el esquema de *Precondición Acción Postcondición* anteriormente expuesto. Sin embargo, la ejecución de estos procedimientos es diferente según tenga la relación la propiedad simétrica o no.

Si la relación no es simétrica, dadas dos relaciones *R* y *R´* tal que *R´* es la relación inversas de *R* y viceversa, y dados dos marcos llamados *Origen* y *Destino*, el esquema seguido para crear las operaciones *Crear_Relación_R*, *Crear_Relación_R´*, *Borrar_Relación_R*, y *Borrar_Relación_R´* es el de la figura 5.54.

Crear_Relación_R (Origen, Destino)

Precondición:
Precondiciones de Crear R

Acción:
Crear_Instancia_Relación_R (Origen, Destino)
Crear_Relación_R' (Destino, Origen)

Postcondición:
Postcondiciones de Crear R

Borrar_Relación_R (Origen, Destino)

Precondición:
Precondiciones de Borrar R

Acción:
Borrar_Instancia_Relación_R (Origen, Destino)
Borrar_Relación_R' (Destino, Origen)

Postcondición:
Postcondiciones de Borrar R

Crear_Relación_R' (Origen, Destino)

Precondición:
Precondiciones de Crear R'

Acción:
Crear_Instancia_Relación_R' (Origen, Destino)
Crear_Relación_R (Destino, Origen)

Postcondición:
Postcondiciones de Crear R'

Borrar_Relación_R' (Origen, Destino)

Precondición:
Precondiciones de Borrar R'

Acción:
Borrar_Instancia_Relación_R' (Origen, Destino)
Borrar_Relación_R (Destino, Origen)

Postcondición:
Postcondiciones de Borrar R'

Figura 5.54 Esquema de la definición de los métodos de las relaciones no simétricas

Pero, si la relación es simétrica, el esquema se simplifica, pues el procedimiento se llama recursivamente al intercambiar el *Origen* por el *Destino,* y viceversa. En la figura 5.55, se muestra el esquema de definición seguido en las relaciones simétricas.

Crear_Relación_R (Origen, Destino)

Precondición:
Precondiciones de Crear R

Acción:
Crear_Instancia_Relación_R (Origen, Destino)
Crear_Relación_R (Destino, Origen)

Postcondición:
Postcondiciones de Crear R

Borrar_Relación_R (Origen, Destino)

Precondición:
Precondiciones de Borrar R

Acción:
Borrar_Instancia_Relación_R (Origen, Destino)
Borrar_Relación_R (Destino, Origen)

Postcondición:
Postcondiciones de Borrar R

Figura 5.55. Esquema de definición en relaciones simétricas

La ejecución de estos procedimientos, suponiendo que se ha solicitado la ejecución de la operación *Crear_Relación_R (Origen, Destino)*, siendo R una relación no simétrica, es la mostrada en la figura 5.56. Si la relación R fuera simétrica, solamente habría que sustituir en la secuencia de ejecución de la figura 5.56, la relación R por R´.

Obsérvese que el esquema propuesto permite:

a) Crear cualquier relación inversa o recíproca asociada a la relación directa.

b) Crear la relación inversa o recíproca y, posteriormente, crear la relación directa.

c) El sistema controla cuándo se debe crear una relación al examinar las precondiciones de los procedimientos.

d) Las postcondiciones garantizan que la relación se ha creado correctamente.

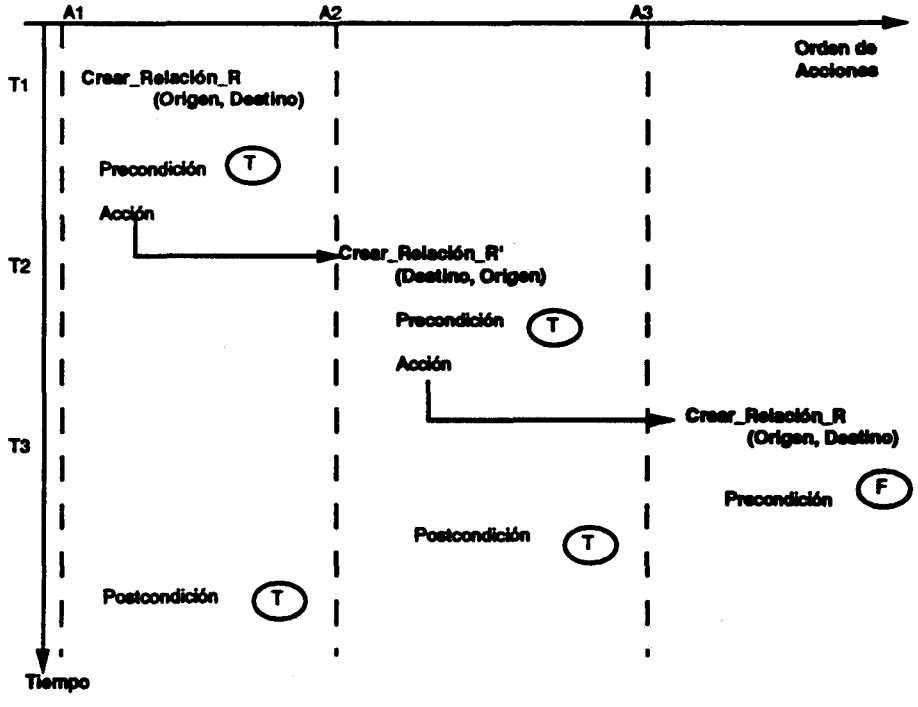


Figura 5.56. Secuencia de acciones realizadas en la ejecución de una relación

Por ejemplo, supóngase la relación SubClase. Se sabe, por la Teoría desarrollada en el apartado 5.2, que esta relación está sujeta a las siguientes restricciones:

1. El origen de la relación es un marco clase,

2. El destino de la relación es un marco clase,

3. No existen circularidades de marcos clase entre el destino y el origen, y

4. No se ha definido previamente esta relación entre el origen y el destino.

Además, se sabe que la relación inversa, relación SuperClase, debe crearse nada más definirse la relación SubClase. Si, en vez de haberse definido primero la relación SubClase se hubiera definido la relación SuperClase, las restricciones a comprobar hubieran sido:

1. El origen es un marco clase,

2. El destino es un marco clase, y

3. No se ha definido previamente esta relación entre el origen y el destino.

y, la relación inversa a crear sería la relación SubClase. Se describe el funcionamiento de estos procedimientos con varios ejemplos.

Ejemplo 1:

Supóngase que se intenta definir entre el marco instanciado *Juan* y el marco clase *Persona* una relación SubClase. Al ejecutarse el procedimiento *Crear_Relación_SubClase(Juan, Persona)*, la primera condición de la precondición asociada a la relación SubClase comprueba que el origen es un marco clase. En este caso, como el origen es un marco instanciado, la primera condición hace a la precondición falsa y no se ejecutan las acciones asociadas a la operación *Crear_Relación_SubClase*.

Ejemplo 2:

Supóngase que se define la relación SubClase entre los marcos clase *Hombre* y *Persona*. Supóngase también que no existen circularidades de ningún tipo entre ambos

marcos y que no se ha definido previamente esta relación entre los dos marcos anteriormente mencionados. Al ser cierta la expresión lógica asociada a la precondición de la operación *Crear_Relación_SubClase(Hombre, Persona)*, se pasa a ejecutar los procedimientos especificados en las acciones de dicha operación.

1. Primero, se crea una instancia del marco *Relación SubClase* con el identificador:

   Hombre_SubClase_Persona

2. A continuación, se pasa a ejecutar el procedimiento que crea la relación inversa, cediendo el control al procedimiento *Crear_Relación_SuperClase (Persona, Hombre)*. Comprobadas la certeza de las precondiciones de este procedimiento, se pasa a ejecutar sus acciones:

   2.1. Se crea una instancia del marco *Relación SuperClase* con el identificador:

   Persona_SuperClase_Hombre

   2.2. A continuación, se solicita la ejecución de la relación inversa asociada a la relación SuperClase, cediendo el control al procedimiento:

   Crear_Relación_SubClase (Hombre, Persona)

   y se pasa a evaluar las precondiciones definidas en este procedimiento. Como la condición cuarta de las precondiciones (no haberse definido previamente la relación) no se cumple en el estado actual, la precondición es falsa. Se sale entonces del procedimiento *Crear_Relación_SubClase* sin haber ejecutado sus acciones y se devuelve el control al procedimiento:

   Crear_Relación_SuperClase (Persona, Hombre)

   Una vez ejecutadas las acciones de dicho procedimiento, se verifica que las postcondiciones son ciertas y se devuelve el control al procedimiento *Crear_Relación_SubClase*.

3. Se ejecuta el procedimiento que crea las relaciones fraternales.

Estos procedimientos se ejecutan al enviar un mensaje a la ranura *Si Añado* o *Si Borro* del marco que representa la relación que se está creando o borrando. En este caso, el procedimiento *Crear_Relación_SubClase* se ejecuta porque se envía un mensaje al marco clase *Relación_SubClase* de la JCM con los parámetros *Hombre* y *Persona*.

La jerarquía de marcos que representa el conocimiento del concepto Relación es la de la figura 5.57. En la figura 5.58, se han representado el conjunto de propiedades y métodos almacenados en el último nivel de la jerarquía de la figura 5.57.

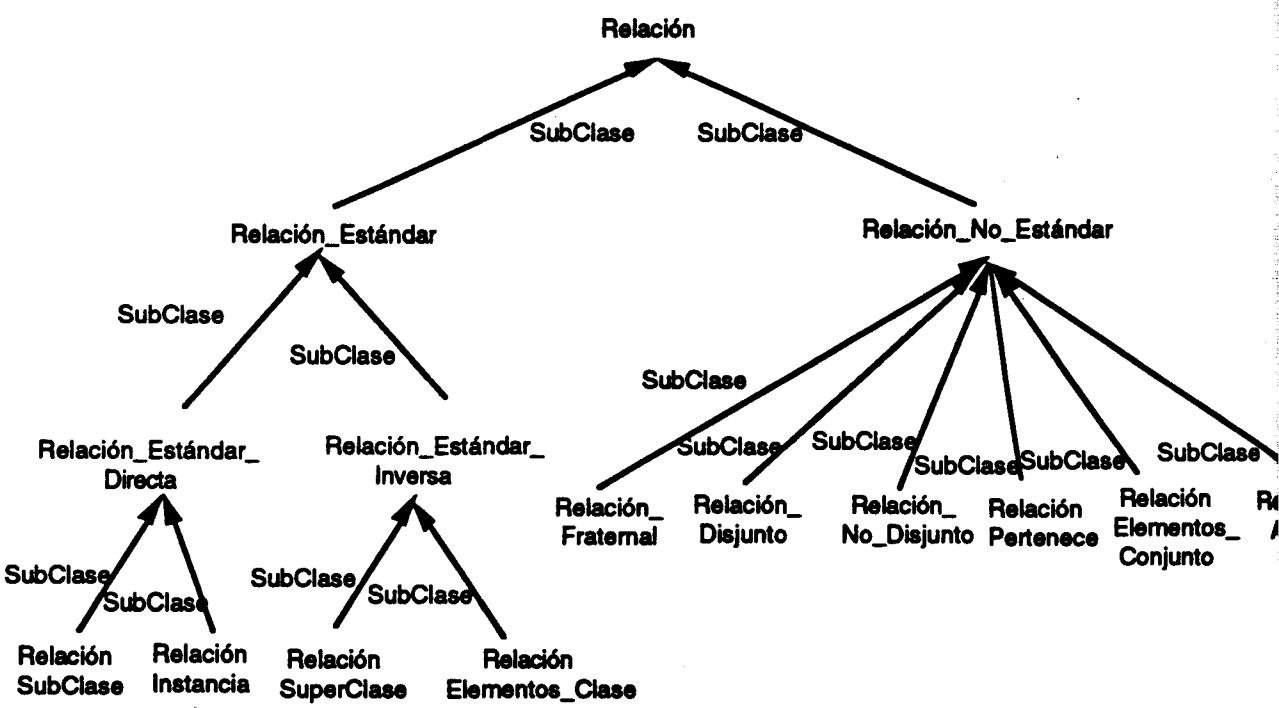


Figura 5.57. Jerarquía de Relaciones.

Las tablas del anexo III comprendidas entre la 5 y la 19 representan a las relaciones como Marcos. En el apartado tercero del mismo anexo, se describe el diseño de cada uno de los procedimientos asociados a las operaciones de *Crear_Relación* y *Borrar_Relación*.

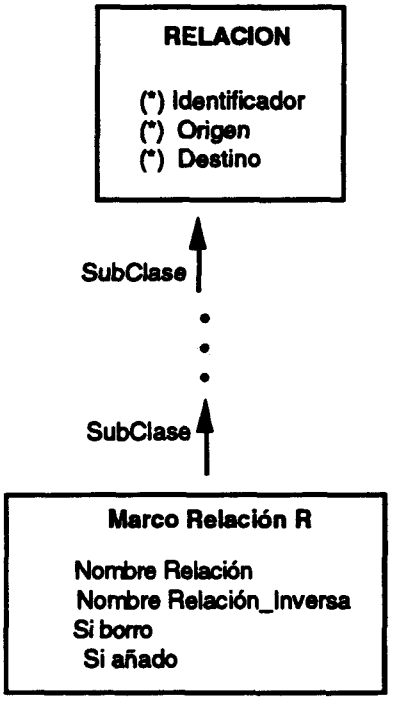


Figura 5.58. Propiedades de los marcos de la Jerarquía de Relaciones

5.4.2.2.2. Propiedades

La jerarquía de marcos que se ha construido en este trabajo para representar los tipos de propiedades que aparecen en los marcos clase, es la de la figura 5.59.

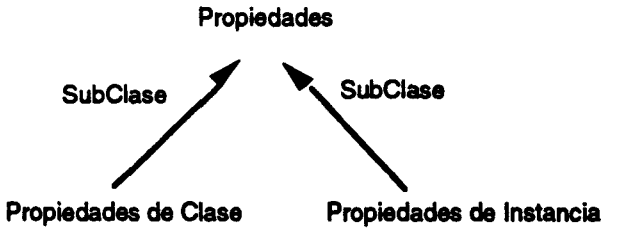


Figura 5.59. Jerarquía de Propiedades

Asociadas a cada uno de estos marcos, se ha definido un conjunto de ranuras, como gráficamente se muestran en la figura 5.60, que representan sus propiedades. Obsérvese que las propiedades de instancia definidas en los marcos *Propiedades*, *Propiedades de Clase* y *Propiedades de Instancia* son las facetas permitidas en cada tipo de propiedad según lo descrito en el apartado 5.2.2.2.2. Las ranuras *Si Añado* y *Si Borro*, son métodos que se ejecutan cuando se solicita crear o borrar una propiedad de un marco concreto de la BC.

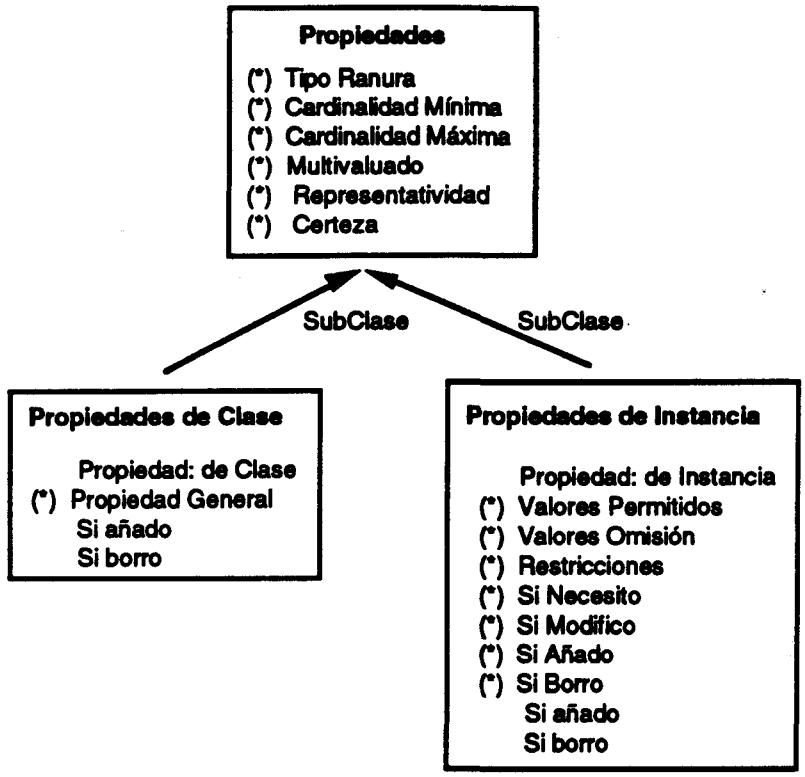


Figura 5.60. Propiedades de Instancia de la Jerarquía de Propiedades.

El esquema que se ha seguido en la construcción de estos procedimientos, es el mostrado en la figura 5.61.

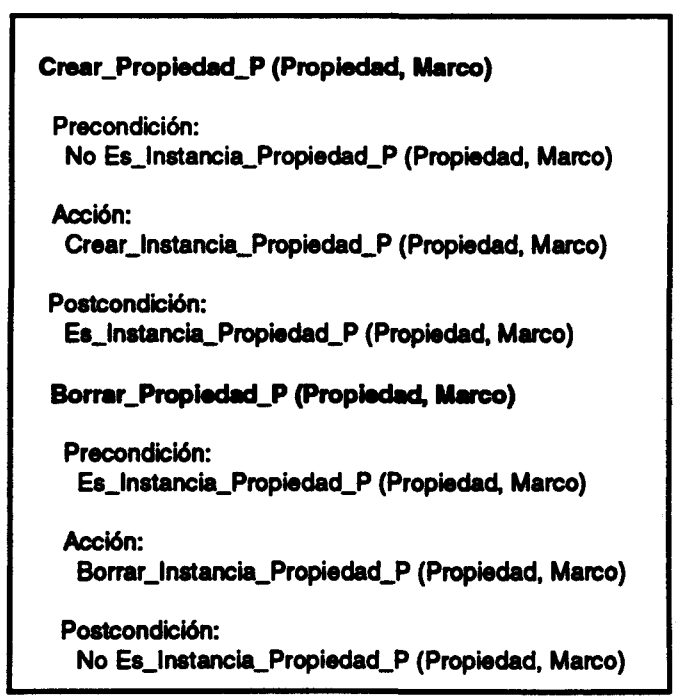


Figura 5.61. Esquema de la definición de los métodos de las propiedades

La ejecución de alguno de estos procedimientos se limita a comprobar que las precondiciones son ciertas y, en caso positivo, crear un marco instanciado del marco *Propiedad_de_Clase* o *Propiedad_de_Instancia*. Automáticamente, si la instancia es una propiedad de Clase, el entorno pedirá al usuario que rellene las propiedades de instancia o facetas asociadas a la propiedad que está definiendo, es decir, que rellene la faceta *Propiedad_General*. Pero, si se tratara de una propiedad de instancia, se rellenarán las propiedades de instancia definidas en el marco clase *Propiedad_de_Instancia*. Se está guiando al IC a :

* Rellenar las facetas permitidas asociadas al tipo de propiedad, y

* Se impide la definición de facetas en propiedades que, por su naturaleza, no las necesitan

Las tablas 20, 21 y 22 del anexo III representan los tipos de propiedades que aparecen en los SS.BB.MM. como Marcos. En el apartado cuarto del mismo anexo, se describen los procedimientos asociados a las operaciones *Crear* y *Borrar* propiedades propias y miembros.

5.4.2.3. Ejemplo

Supóngase que el IC desea construir la jerarquía de la figura 5.62 en este entorno.

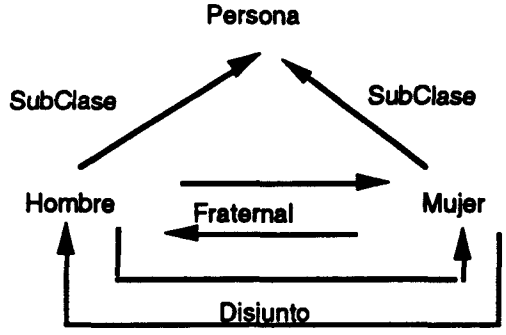


Figura 5.62. Ejemplo de una Jerarquía construida por el IC

La representación interna que utiliza el entorno para representar este conocimiento es la mostrada en la figura 5.63. Obsérvese que:

a) Los conceptos *Persona*, *Hombre* y *Mujer* son instancias en la JCM del marco *Marco Clase*.

b) Las relaciones *SubClase* entre *Hombre* y *Persona*, y entre *Mujer* y *Persona* se representan internamente como dos marcos instanciados llamados *Hombre_SubClase_Persona* y *Mujer_SubClase_Persona*. Estos marcos son instancias del marco *Marco Relación_SubClase* de la JCM. Desde las ranuras *SubClase* definidas en los marcos *Hombre* y *Mujer* parten punteros a dichos marcos instanciados.

c) Las relaciones *Disjunto* entre *Hombre* y *Mujer* se representan internamente como dos marcos instanciados llamados *Hombre_Disjunto_Mujer* y *Mujer_Disjunto_Hombre*. Estos marcos son instancias del marco *Marco Relación_Disjunto* de la JCM. Desde las ranuras *Disjunto* definidas en los marcos *Hombre* y *Mujer* parten punteros a dichos marcos instanciados.

d) Igual sucedería para las relación *Fraternal* definida entre *Hombre* y *Mujer*

5.4.3. RAZONAMIENTO

Se han implementado los algoritmos que realizan la Equiparación, la Herencia de Propiedades y Donación de Propiedades descritos en el apartado 5.4. Estos algoritmos trabajan sobre las estructuras que se acaban de describir en el apartado 5.4.2.

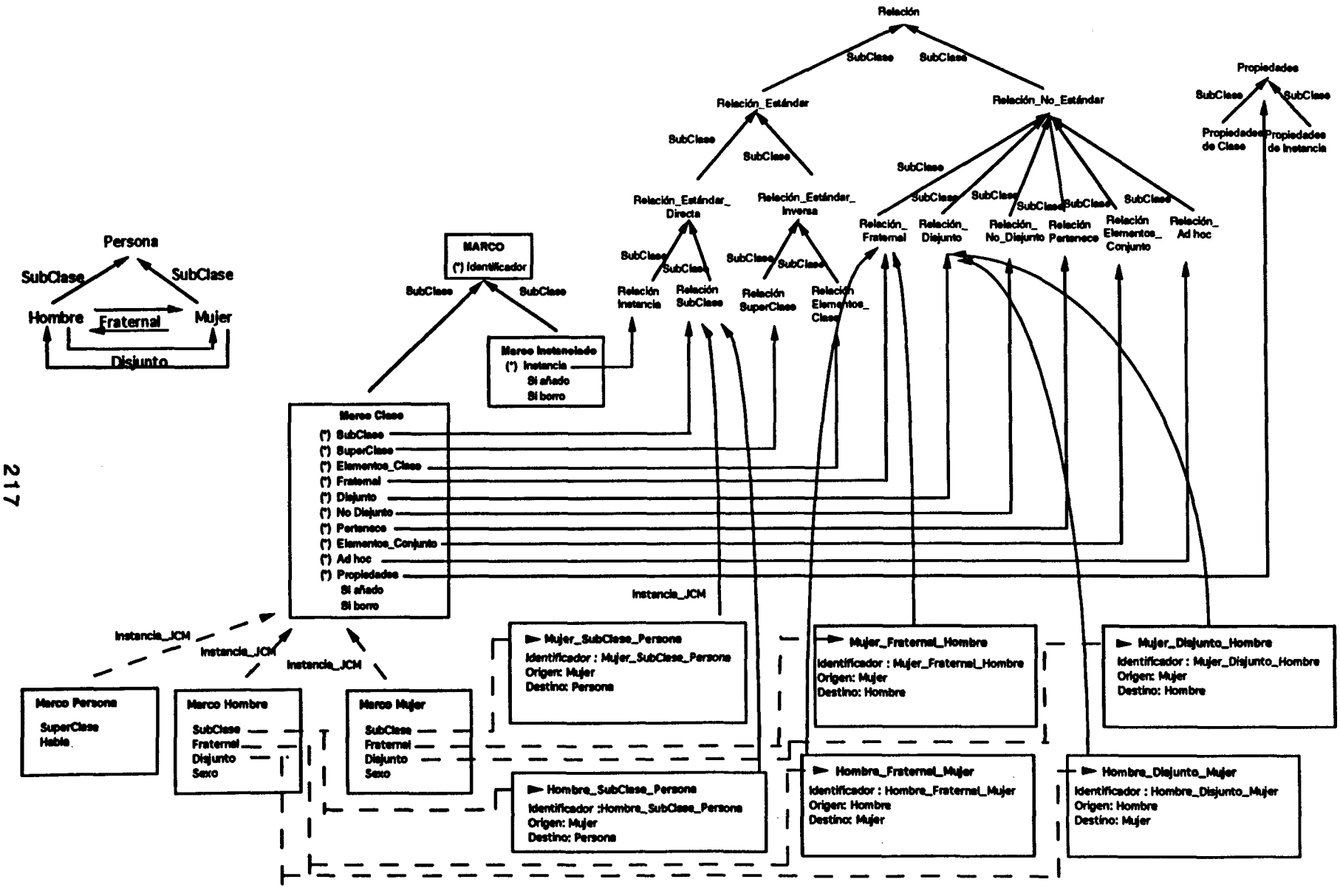


Figura 5.63 Representación de una jerarquía en el entorno

6. CONCLUSIONES

6. CONCLUSIONES

La aportación teórica de este trabajo consiste en formalizar el paradigma de Marcos en un Modelo de Diseño. El **Modelo de Diseño Orientado a Marcos** permite construir *Modelos Formalizados* de la realidad, intermedios entre el Mundo Externo y el Modelo Interno del mundo que supone el sistema implementado, disminuyendo así las pérdidas de conocimiento que se producen al formalizar BC utilizando Marcos. En concreto:

a) Se ha definido con rigurosidad la terminología, la sintáxis y la semántica de los conceptos que aparecen en el paradigma, y se han establecido un conjunto de restricciones sintácticas y semánticas que impiden al IC definir en su *Modelo Formalizado* de la realidad elementos no permitidos en el **Modelo de Diseño**, o el uso indebido de ellos. Se evitarán así las incoherencias sintácticas y lógicas en la BC.

b) Se ha incrementado la expresividad del formalismo de Marcos al introducir en el **Modelo de Diseño Orientado a Marcos**:

   b.1) La *Representatividad* de las propiedades en los marcos clase.

   b.2) Grados de certeza de las propiedades en las entidades.

   b.3) Un conjunto de relaciones no estándar estructurales proporcionadas y gestionadas por el **Modelo de Diseño** (Fraternal, Disjunta y No Disjunta).

   b.4) Un conjunto de relaciones no estándar definidas a medida por el usuario y gestionadas por el **Modelo de Diseño.**

c) Se han incrementado la eficacia y la eficiencia de las inferencias al:

   c.1) Definir de un modo preciso los mecanismos de inferencia clásicos de razonamiento en marcos: Equiparación y Herencia.

   c.2) Modificar e incorporar técnicas de razonamiento que han hecho más potente, y han acercado más la realidad, al paradigma de representación de los Marcos. Concretamente:

c.2.1) Se ha modificado la Equiparación, que trabaja con el conocimiento incierto del dominio y con la representatividad de las propiedades conocidas. La representatividad permite al SBM realizar inferencias correctas con información incompleta cuando la representatividad de una propiedad en una clase toma el valor uno y la certeza de dicha propiedad en la entidad también.

c.2.2) Se ha introducido la Cesión de propiedades basada en *Donación* que, combinada o no con la técnica de herencia, permite incrementar aún más la compartición y distribución de las propiedades en los SS.BB.MM.

d) El análisis de las propiedades y facetas ha llevado a proponer una tabla en el **Modelo de Diseño** como medio de expresión de los marcos utilizado por el IC al construir el *Modelo Formalizado*.

Por otro lado, la aportación de este trabajo, desde un punto de vista práctico, consiste en la construcción de un entorno de formalización de SS.BB.MM., en el que todas las aportaciones teóricas propuestas han sido desarrolladas. Este entorno, es un SBM formalizado utilizando los conceptos propuestos y definidos en el **Modelo de Diseño** del apartado 5.2 y 5.3. Se trata, pues, de un SBM que permite al IC formalizar BC basadas en Marcos y construir *Modelos Formalizados* de la realidad y soportados por el **Modelo de Diseño** que se propone.

Todas estas características permiten afirmar que el **Modelo de Diseño** de representación propuesto:

* Es **independiente del problema**. El IC puede utilizar el **Modelo de Diseño** para formalizar cualquier BC en cualquier dominio, es decir, para construir su *Modelo Formalizado* de la realidad.

* Es **flexible**. Dada la modularidad del modelo es fácil introducir y gestionar nuevas relaciones o nuevos conceptos en él.

* Facilita el **desarrollo incremental** en el conocimiento y en las inferencias del sistema inteligente al permitir al IC razonar en cada momento con el conocimiento almacenado en su *Modelo Formalizado*.

- Permite la **evaluación de las respuestas** del sistema en la etapa de formalización, en lugar de hacerlo en la etapa de implementación. Por tanto, no es necesario esperar a que el sistema esté implementado para verificarlo y validarlo.

7. FUTURAS LINEAS DE INVESTIGACION

7. FUTURAS LINEAS DE INVESTIGACION

A modo de continuación, ampliación y profundización del Modelo de Diseño del presente trabajo de investigación, se sugieren las siguientes futuras líneas:

1. *Elaborar un procedimiento independiente del dominio que utilice los valores activos y los métodos para realizar inferencias.*

   La ejecución de los procedimientos asociados a las facetas procedimentales de los marcos presentan las siguientes características:

   a) Los procedimientos se definen en el marco clase y permanecen latentes a menos que solicite su ejecución.

   b) Los procedimientos se ejecutan asíncronamente cuando ciertos hechos ocurren sobre los datos almacenados en las propiedades de los marcos instanciados de la BC.

   c) El control va pasando de unas ranuras a otras a medida que se van realizando inferencias con los datos almacenados en la BC.

   d) Un procedimiento puede ceder el control a otro procedimiento. Una vez ejecutado éste, el control vuelve al procedimiento inicial, procedimiento que sigue ejecutándose en el mismo lugar en el que se detuvo.

   e) Los procedimientos pueden simular encadenamiento hacia delante y así tener demonios dirigidos por los eventos, y encadenamiento hacia atrás y tener encadenamiento dirigido por las metas. El primer tipo de demonio, es el asociado a las facetas si se añade, si se modifica y si se borra. El segundo, es el asociado a la faceta si se necesita. La figura 7.1, muestra para cada una de las facetas, cuales son el conjunto de facetas accesibles desde ella.

   f) Es necesario idear un sistema que evite la presencia de bucles y que sea independiente del dominio.

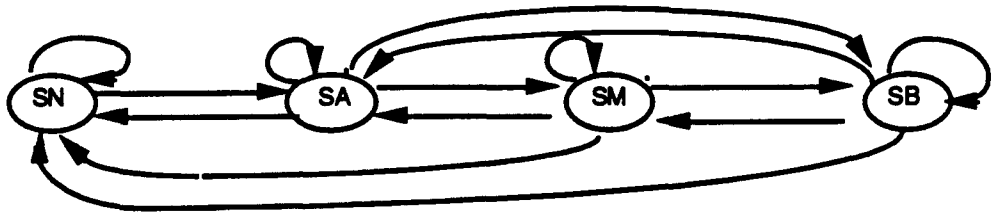


Figura 7.1. Llamadas de Valores Activos

2. *Construcción de un método que ayude al IC a formalizar BC basadas en Marcos.*

a) El método debe ayudar al IC a:

* Seleccionar los conceptos más significativos del dominio,
* Definir las relaciones "ad hoc" entre dichos conceptos,
* Construirse las jerarquías para cada uno de estos conceptos,
* Distribuir inteligentemente las propiedades en cada marco de tal forma que se obtenga un equilibrio entre el número de propiedades definidas y la información que cada una de ellas aporta al marco.
* Determinar el marco más adecuado para definir la propiedad.

b) Además, en dicho método se deberían utilizar técnicas basadas en la Teoría de la Información que determinen la cantidad de información que proporciona cada propiedad al marco en el que está definida y cada marco al global de la BC.

3. *Introducción de técnicas borrosas en el Modelo.*

Se realizaría mediante:

a) El uso de definiciones borrosas de las propiedades.

b) Definiciones borrosas de la representatividad de las propiedades.

c) La asignación de valores de certeza obtenidos por técnicas borrosas a las certezas de las propiedades de una entidad.

d) La modificación de las inferencias basadas en equiparación y cesión de propiedades para que trabajen con este tipo de valores.

4. *Aplicar aprendizaje inductivo a la técnica de equiparación.*

Utilizar la técnica de aprendizaje inductivo ID3 para elaborar un conjunto de reglas que guien el proceso de selección del conjunto de marcos candidatos con los que una entidad puede equipararse.

8. BIBLIOGRAFIA

[Alonso, 93] Alonso, F. *Unidad Didáctica de Entornos de Programación.* **Máster en Ingeniería del Software.** Facultad de Informática. Universidad Politécnica de Madrid. Madrid. España. 1993.

[Alpert et al., 90] Alpert, S.R.; Woyak, S.W.; Shrobe H.J.; Arrowood, L.F. *Object-Oriented Programming in AI.* **IEEE Expert.** Diciembre, 1990. pp:6-7.

[America, 87] America, P. *Inheritance and Subtyping in a Parallel Object-Oriented Language.* En Bézivin, J; Hullot, J.M.; Cointe, P y Lieberman, H. **ECOOP'87: European Conference on Object-Oriented Programming.** Paris, Francia, 15 al 17 de Junio, 1987. pp: 234-42.

[America, 89] America, P. *A Behavioural Approach to Subtyping.* En Lenzerini, M.; Nardi, D.; Simi, M. **Inheritance Hierarchies in Knowledge Representation and Programming Languages.** John Wiley & Sons. Nueva York, EE.UU. 1991. pp: 173-90.

[Barlett, 32] Barlett F.; **Remembering: A Study in Experimental and Social Psychology.** Cambridge University Press. Cambridge, U.K. 1932.

[Barr et al., 82] Barr, A.; Feigenbaum, E.A. **The Handbook of Artificial Intelligence.** Vol. 1. Addison Wesley. Reading, MA, EE.UU. 1982.

[Belnap,77] Belnap, N.; *A Useful Four-Valued Logic.* En Epstein. D. Reidel, Dordrecht. **Modern Uses of Multiple-Valued Logic.** Dunn, J.M.; 1977, pp: 8-37.

[Block et al., 89] Block, F.P.; Chan, N.C. *An Extended Frame Language.* **OOPSLA'89 Proceedings.** Octubre, 1989. pp: 151-7.

[Blum, 92] Blum, B.I. *The Evolution of Software Engenieering.* **First International Conference of Information and Knowledge Management.** Baltimore, MD, EE.UU. 8 al 11 de Noviembre 1992.

[Bobrow et al., 77] Bobrow D. G., Winograd T; *An Overview of KRL, a Knowledge. Representation Language.* **Cognitive Science.** 1977. pp: 3-46.

[Bobrow et al., 88] D. Bobrow et al., *Common LISP Object System Specification X3J13.* **SIGPlan Notices,** Septiembre, 1988.

[Bobrow et al., 90] Bobrow, D.G.; Demichiel, L.G.; Gabriel, R.P.; Keene, S.E.; Kiczales, G; Moon, D.A. **Common Lisp Object System: Common Lisp. The Language.** Steele JR. G.L. Digital Press. 1990.

[Borrajo et al., 93] Borrajo, D.; Juristo, N.; Martínez, V.; Pazos, J. **Inteligencia Artificial. Métodos y Técnicas.** Ed. Ceura. Madrid, España. 1993.

[Brachman, 79] Brachman R.J.; *On the Epistemological Status of Semantic Networks.* En N. V. Findler. **Associative Networks.** Academic Press. Nueva York, EE.UU. 1979.

[Brachman et al., 85] Brachman, R.J.; Gilbert, V.P.; Levesque, H.J. *An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON.* **Proceedings IJCAI.** 1985.

[Breuker et al., 85] Breuker, J.; Wielinga, B. *KADS: Structured Knowledge Acquisition for Expert Systems.* **5th International Workshop on Expert Systems and their Applications.** Agence de l'Informatique. Avignon, Francia. 1985.

[Buchanan et al., 84] Buchanan. B.G.; Shortliffe, E.H. **Rule-Based Expert Systems: The MYCIN Experiments.** Addison-Wesley. Reading, MA, EE.UU.1984.

[Caraça et al., 93] Caraça, J.P.; Gómez-Pérez, A.; Juristo, N. y Pazos, J. *The Serendipitous Effect and Artificial Intelligence.* **5th UNB Artificial Intelligence Symposium.** Fredericton, N.B. Canadá.11-14 de Agosto, 1993. pp: 309-19.

[Coad et al., 91] Coad, P.; Yourdon, E. **Object Oriented Design.** Yourdon Press. Prentice Hall. Englewood Cliffs. NJ, EE.UU. 1991.

[Dahl et al., 66] Dahl, O.J.; Nygaard, K. *Simula: an ALGOL-based Simultion Language.* **Communications of the ACM.** Vol. 9; Septiembre, 1966. pp: 671-8.

[Dahl et al., 70] Dahl, O-J; Myhrhaug, B.; Nygaard, k. **The Simula 67 Common Base Language.** Publication S22, Norwegian Computing Centre, Oslo, 1970.

[Davis et al., 93] Davis, R; Shrobe, H.; Szolovits, P. *What is a Knowledge Representation?.* **AI Magazine.** Spring, 1993. pp:17-33.

[De Francesco, 93] De Francesco, M. *Frames vs. Objects* **Article: 16112 of comp.ai.** 24, marzo, 1993.

[De Millo et al., 79] De Millo, R.A.; Lipton, R.J.; Perlis, A.J. *Social processes and proofs of theorems and programs.* **Communication of the ACM.** 22, Mayo 1979. pp: 535-8.

[Dijkstra, 76] Dijkstra, E.W. **A Discipline of Programming.** Prentice Hall, Englewood Cliffs. NJ, EE.UU, 1976.

[Doyle, 79] Doyle, J. *A Truth Maintenance System.* **Artificial Intelligence.** Vol. 12, No. 3, 1979.

[Etherington et al.,83] Etherington, D.W.; Reiter, R. *On Inheritance Hierarchies with Exceptions.* **Proceedings of AAAI-83.** 1983. pp: 104-8.

[Etherington, 87] Etherington, D.W. *Formalizing Nonmonotonic Reasoning Systems.* **Artificial Intelligence.** Vol. 31, 1987. pp: 41-85.

[Fahlman, 79] Fahlman, S.E. **NETL: A System for Representating and Using Real-World Knowledge.** The MIT Press, Cambridge, MA, EE.UU. 1979.

[Fikes et al., 85] Fikes R.; Kehler, T. *The Role of Frame-Based Representation in Reasoning.* **Communications of the ACM.** Vol 28. Nº 9. Septiembre 1985. pp: 904-20.

[Floyd, 67] Floyd, R. *Assigning Meaning to Programs.* **Mathematical Aspects of Computer Science.** XIX American Mathematical Society. 1967. pp 19-32.

[Frost, 86] Frost, R.A. **Introduction to Knowledge Based Systems.** MacMillan. Nueva York, EE.UU. 1986.

[Goldberg et al., 83] Goldberg, A.; Robson, D. **Smalltalk-80: The Language and its Implementation.** Addison-Wesley, Reading, MA, EE.UU. 1983.

[Gómez, 93] Gómez, A. **Estudio de Técnicas de Representación del Conocimiento.** Tesis de Máster en Ingeniería del Conocimiento. Facultad de Informática. Universidad Politécnica de Madrid. Madrid. España. 1993.

[González et al., 93] González, J.A.; Dankel, D.D. **The Engineering of Knowledge-Based Systems. Theory and Practic.** Prentice Hall. Englewoods Cliffs, NJ, EE.UU. 1993.

[Gries, 81] Gries, D. **The Science of Programming.** Springer-Verlag. Nueva York, EE.UU. 1981.

[Hoare, 69] Hoare, C.A.R. *An axiomatic Approach to Computer Programming.* **Communication of the ACM,** 12. Octubre 1969. pp: 576-80.

[Hoare, 72] Hoare C.A.R. *Proofs of Correctness of Data Representation.* **Acta Informática.** Vol. 1. 1972. pp: 271-81.

[Horty et al., 87] Horty, J.F.; Thomason, R.H.; Touretzky, D.S. **A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks.** Department of Computer Science. Carnegie-Mellon University. CMU-CS-87-175. Pittsburgh, PA, EE.UU. 1987.

[Horty, 89] Horty, J.F. *A Credulous Theory of Mixed Inheritance.* En Lenzerini, M.; Nardi, D.; Simi, M. **Inheritance Hierarchies in Knowledge Representation and Programming Languages.** John Wiley & Sons. Nueva York, EE.UU. 1991. pp: 13-28.

[Ibrahim et al., 91] Ibrahim, M.; Bobrow, D.; Hewitt, c.; Perror, J.F.; Smith, R.; Shrobe, H.; *OOP and AI (PANEL).* **OOPSLA'91.** 1991. pp: 351-4.

[Jacobson, 93] Jacobson, I. *Is Object Technology Software's Industrial Platform?.* **IEEE SOFTWARE.** Enero, 1993, pp: 24-30.

[Khoshafian et al., 90]Khoshafian, S.; Abnous, R. **Object Orientation: Concepts, Language, Databases, User Interfaces.** John Wiley & Sons, Inc. Nueva York, EE.UU. 1990.

[Ledbetter et al., 85] Ledbetter, L.; Cox, B. *Software-IC's.* **BYTE.** Junio, 1985.

[Lenat et al., 90] Lenat, D.B.; Guha, R.V. **Building large Knowledge-Based Systems: Representation and Inference in the CYC Project.** Addison Wesley. Reading, MA, EE.UU. 1990.

[Lenzerini et al., 91] Lenzerini, M.; Nardi, D.; Simi, M. **Inheritance Hierarchies in Knowledge Representation and Programming Languages.** John Wiley & Sons. Nueva York, EE.UU. 1991.

[McCarthy, 61] McCarthy, J. *A Basis for a Mathematical Thory of Computation.* **Proceedings Western Joint Comp. Conf.** Los Angeles, EE.UU. Mayo, 1961. pp: 225-38.

[McDermott et al., 80] McDermott, D.; Doyle, J. *Non-Monotonic Logic.* **Artificial Intelligence.** Vol. 13. Abril, 1980. pp: 41-72.

[Meyer, 88] Meyer, B. **Object-Oriented Software Construction.** Prentice Hall. Nueva York, EE.UU. 1988.

[Minsky, 75] Minsky M.; *A Framework for Representating Knowledge.* En P. Winston. **The Psychology of computer Vision.** McGraw-Hill Nueva York, EE.UU. 1975.

[Mitchell et al., 89] Mitchell, T.M.; Allen, J.; Chalasani, P.; Cheng, J.; Etzioni, M.; Ringuette, M.; Schlimmer, J. *THEO: A Framework for Self-improving Systems.* En K. VanLehn **Architectures for Intelligence.** Erlbaum. Hillsdale, NJ, EE.UU. 1989.

[Molina, 93] Molina, M. **Desarrollo de Aplicaciones a Nivel Cognitivo Mediante Entornos de Conocimiento Estructurado.** Tesis Doctoral. Departamento de Inteligencia Artificial. Facultad de Informática. Universidad Politécnica de Madrid. Madrid, España. 1993.

[Moon, 86] Moon, D.A. *Object Oriented Programming with Flavors.* **Proceedings of OOPSLA-86,** Portland, Oregon, EE.UU. 1986.

[Nado et al., 87] Nado, R.; Kikes, R. *Semantically Sound Inheritance for a Formally Defined Frame Languaje with Defaults.* **Proceedings of de AAAI-87.** 1987. pp: 443-8.

[Nado et al., 92] Nado, R; Fikes, R. *Saying More Whit Frames: Slots as Classes.* **Computer Mathematics Application.** Vol. 23, Nº 6-9, 1992. pp: 719-31.

[Naur, 66] Naur, P. *Proofs of Algorithms by General Snapshots.* **BIT** 6. 1966. pp: 299-316.

[Newell, 82] Newell, A. *The Knowledge Level.* **Artificial Intelligence.** Vol. 18. 1982. pp: 87-127.

[Patel-Schneider, 91] Patel-Schneider, P.F. *What's Inheritance Got to do with Knowledge Representation.* En Lenzerini, M.; Nardi, D.; Simi, M. **Inheritance Hierarchies in Knowledge Representation and Programming Languages.** John Wiley & Sons. Nueva York, EE.UU. pp: 1-12. 1991.

[Quillian, 68] Quillian, R. *Semantic Memory*, En M. Minsky. **Semantic Information Processing.** MIT Press, Cambridge, MA, EE.UU. 1968.

[Reichgelt, 91] Reichgelt H.; **Knowledge Representation: An AI Perspective;** Ablex Publishing Corporation, Norwood. NJ, EE.UU. 1991.

[Reiter, 78] Reiter, R.; *On closed World Data Bases.* En Gallaire, H.; Minker, J. **Logic and Data Bases.** Plenum Press. Nueva York, EE.UU. 1978. pp: 55-76.

[Reiter, 80] Reiter, R.; *A Logic for Default Reasoning.* **Artificial Intelligence** Vol. 13. 1980. pp: 81-132.

[Rich, 83] Rich, E. **Artificial Intelligence.** McGraw-Hill. Nueva York, EE.UU. 1983.

[Rich et al., 91] Rich E., Knight K.; **Artificial Intelligence;** McGraw-Hill: Nueva York, EE.UU. Segunda Edición. 1991.

[Roberts, 92] Roberts, R.M. **Serendipia.** Alianza Editorial, S.A. Madrid. España. 1992.

[Roberts et al., 77] Roberts, R.B.; Goldstein, I.P. **The FRL Manual.** A.I. Memo Nº 409, MIT Artificial Intelligence Laboratory. Reading, MA, EE.UU. 1977.

[Roche, 89] Roche, C. *From Objects to Frames: Artificial Intelligence Knowledge Representation in Smalltalk 80.* **TOOLS'89.** 1989. pp: 177-85.

[Schaffert et al., 86] Schaffert. C.; Cooper, Y.; Bullis, B.; Kilian, M.; Wilpolt, C. *An Introduction to Trellis/Owl.* **OOPSLA-86.** 1986. pp: 9-16.

[Schank, 72] Schank R.; *Conceptual Dependency: a theory of Natural Language Understandig.* **Cognitive Psychology**, 3, 552-631. 1972

[Schank, 75] Schank R.; *The structure of episodes in memory.* en Bobrow, D.G. y Collins (Eds). **Representation and Understandig. Studies in congnitive science.** New York: Academic Press. 1975

[Schank et al., 77] Schank R., Abelson R.; **Scripts, plans, goals and understanding: An inquiry into human knowledge structures.** Hillsdale, NJ: Lawrence Erlbaum Associates. 1977

[Schmucker, 86] Schmucker, K.J. **Object-Oriented Programming for the Macintosh.** Hayden Book Co. Hasbrouck Heights, NJ, EE.UU. 1986.

[Selman et al., 89] Selman, B.; Levesque H.J. *The Tractability of Path-based Inheritance.* En Lenzerini, M.; Nardi, D.; Simi, M. **Inheritance Hierarchies in Knowledge Representation and Programming Languages.** John Wiley & Sons. Nueva York, EE.UU. 1991. pp: 83-96

[Smith, 82] Smith, B.; **Reflection and Semantics in a Procedimental Language.** Tesis Doctoral, MIT, Cambridge, MA, EE.UU. 1982.

[Snyder, 86] Snyder, A. *Encapsulation and Inheritance in Object-Oriented Programming Languages.* **Proceedings of the ACM Conference on Object Oriented Programming, Systems, Languages and Applications.** Portland, Oregon, EE.UU. Septiembre 1986. pp: 9-16.

[Snyder, 89] Snyder, A. *Inheritance in Object-Oriented Programming Language.* En Lenzerini, M.; Nardi, D.; Simi, M. **Inheritance Hierarchies in Knowledge Representation and Programming Languages.** John Wiley & Sons. Nueva York, EE.UU. 1991.

[Snyder, 93] Snyder, A. *The Essence of Objects: Concepts and Terms.* **IEEE SOFTWARE**. Enero, 1993. pp:31-42.

[Steels, 90] Steels, L. *Componente of Expertise.* **AI Magazine.** Vol. 11. (2). 1990. pp:

[Stefik, 79] Stefik, M. *An Examination of a Frame-Structured Representation System* . **Proceedings IJCAI.** 1979. pp: 845-52.

[Stefik et al., 86] Stefik, M.; Bobrow D. *Object Oriented Programming: Themes and Variations.* **AI Magazine.** Winter, 1986. pp: 40-62.

[Stein, 89] Stein, L.A. *Computing Skeptical Inheritance.* En Lenzerini, M.; Nadi, D.; Simi, M. **Inheritance Hierarchies in Knowledge Representation and Programming Languages.** John Wiley & Sons. Nueva York, EE.UU. 1991. pp: 69-82.

[Steinheiser, 90] Steinheiser, F.H. *Frame-Based Systems.* En White, M.; Goldsmith, J. **Standards and Review Manual for Certification in Knowledge Engineering.** IAKE. Washington, DC, EE.UU. 1990.

[Stroustrup, 86] Stroustrup, B. **The C++ Programming Language.** Addison-Wesley. Reading, MA, EE.UU. 1986.

[Thomason et al., 86] Thomason, R.H.; Horty, J.F.; Touretzky D.S. **A Calculus for Inheritance in Monotonic Semantic Nets.** Department of Computer Science. Carnegie Mellon University. CMU-CS-86-138. Pittsburgh, PA, EE.UU. 1986.

[Touretzky, 84] Touretzky, D.S. **The Mathematics of Inheritance Systems.** CMU-CS-84-136. Department of Computer Science. Carnegie Mellon University. Pittsburgh, PA, EE.UU. 1984.

[Touretzky, 86] Touretzky, D.S. **The Mathematics of Inheritance Systems.** Morgan Kaufmann Publishers, Inc., Los Altos, CA, EE.UU. 1986.

[Touretzky et al., 87] Touretzky, D.S.; Horty J.F.; Thomason R.H. *A Clas of Intuitions: The Current State of Nonmonotonic Multiple Inheritance Systems.* **Proceedings of the Tenth International Joint Conference on Artificial Intelligence.** Vol 1. 1987. pp: 476-82.

[Turing, 50] Turing, A.M. *Computing Machinery and Intelligence.* **Mind.** Vol. LIX; Nº 236; 1950.

[Walters et al., 88] Walters, J.R.; Nielsen, N.R. **Crafting Knowledge-based Systems: Expert Systems Made Realistic.** Wiley-Interscience. Nueva York, NY, EE.UU. 1988.

[Winblad et al., 90] Winblad, A.L.; Edwards, S.D.; King, D.R. **Object Oriented Software.** Addison-Wesley. Publishing Company, Inc. Reading, MA, EE.UU. 1990.

[Winograd, 75] T. Winograd; *Frame Representation and the Declarative/Procedimental Controversy.* En Bobrow, D.G.; Collins, A.M. **Representation and Understanding: Studies in Cognitive Science.** Academic Press. Nueva York, EE.UU. 1975. pp: 185-210.

[Winston, 92] Winston, P.H. **Artificial Intelligence.** Addison Wesley. Reading, MA, EE.UU. 1992.

[Woods, 91] Woods, W. A. *Important Issues in Knowledge Representation.* **Proceedings of the IEEE.** Vol. 74. Nº 10. Octubre 1991. pp: 146-58.

ANEXO I.
EJEMPLO DE EQUIPARACION

Operador_División_Entera, *Operador_Multiplicación* y *Operador_Menor_Igual* respectivamente.

Por ejemplo, supóngase que se desea realizar la ejecución de la operación lógica anterior, donde *a, b, c, d, e y f* son variables que representan números enteros. Cuando por cualquier motivo se solicite la ejecución de la operación representada por el marco *Operador_Y_1*, la ranura *Ejecutar* del marco instanciado en cuestión recibe un mensaje de activación con los valores de las variables *a, b, c, d, e y f*. Estos valores pueden ser: 1, 2, 3, 4, 5 y 6. Como la ranura *Ejecutar* no se ha definido en el marco instanciado *Operador_Y_1*, aplicando los mecanismos de herencia, se accede al marco clase *Operador_Y*. En él se encuentra definida la ranura buscada y comienza a ejecutarse el procedimiento *Y (Operador1, Operador2, Resultado)* instanciado:

Y ([(a Div b) > (c*d)], (e =< f), Resultado).

Como el *Operando1* de la operación Y representa la operación compuesta "[(a Div b) > (c*d)]" y el *Operando2* la operación simple "(e =< f)", se pasa a ejecutar cada una de las operaciones por separado. Primero se comienza a ejecutar la operación almacenada en el *Operando1* y se envía un mensaje a la ranura *Ejecutar* del marco instanciado *Operador_Mayor_1* solicitando la ejecución de la operación instanciada siguiente:

Mayor ((a Div b), (c*d), Resultado).

Como en el marco instanciado *Operador_Mayor_1* no se encuentra la ranura *Ejecutar*, aplicando herencia, se accede al marco clase *Operador_Mayor*, encontrándose la ranura y comenzando la ejecución de la operación. Al no ser los operandos que intervienen en la operación Tipos Básicos sino operaciones, se pasa a ejecutar cada una de las operaciones instanciadas. Primero se envía un mensaje al marco instanciado *Operador_División_Entera_1* con los parámetros simples *a* y *b* con los que puede ejecutar directamente el procedimiento:

División_Entera (a, b, Resultado)

que almacena en la ranura *Resultado* el resultado de la operación. La ejecución de la operación se realiza como sigue: el marco clase *Operador_División_Entera* envía un mensaje al objeto *a* de la *Jerarquía de Tipos* con el operando *División_Entera* y con *b* como parámetro, ejecutándose la operación *a Div b* físicamente en la computadora y

EJEMPLO DE EQUIPARACION

En este anexo se muestran varios ejemplos de equiparación de una entidad E_i con los marcos clase de la jerarquía de vertebrados mostrada en la figura 5.18.

A lo largo de todos los ejemplos se observa cómo, a medida que se va añadiendo más propiedades conocidas a la entidad, la clase con la que se equipara es más consistente con la descripción dada.

Cada ejemplo de equiparación se resume en una tabla. El esquema que a continuación se propone en la tabla I.1 ha sido el esquema seguido en todos ellos.

	Propiedades Conocidad en la Entidad						
	C(r) C(c)	D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) D2(c)	C(r) D2(c)	D2(r) D2(c)
MC_1	VE_1	VE_2	VE_3	VE_4	VE_5	VE_6	VE_7
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
•	•	•	•	•	•	•	•
MC_n	VE'_1	VE'_2	VE'_3	VE'_4	VE'_5	VE'_6	VE'_7

Tabla I.1.

donde:

C (r): Conocida la Representatividad de las propiedades en la clase

C (c): Conocida la Certeza de las propiedades en la entidad

D1 (r): Desconocida la Representatividad. Valor por Omisión: 0,001

D1 (c): Desconocida la Certeza. Valor por Omisión: 0,001

D2 (r): Desconocida la Representatividad. Valor por Omisión: 0,2

D2 (c): Desconocida la Certeza. Valor por Omisión: 0,2

MC ¡: Marco Clase i

VE ¡: Valor de Equiparación i

E1	Esqueleto C(r) C(c)	Sí D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Vertebrado	1	0.0001	0.001	0	0.2	0.2	0.04

Tabla I.2

E2	Esqueleto Respiración C(r) C(c)	Sí Pulmonar D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Vertebrado	1	0.0001	0.001	0	0.2	0.2	0.04
Mamífero	0.65	0.0001	0.0008	0	0.28	0.154	0.0592
Reptiles	0.65	0.0001	0.0008	0	0.28	0.154	0.0592
Aves	0.6	0.0001	0.0007	0	0.28	0.136	0.0592
Anfibio	0.55	0.0001	0.0006	0	0.28	0.118	0.0592

Tabla I.3

E3	Esqueleto Respiración C(r) C(c)	Sí Cavidad_Bucal D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Vertebrado	1	0.0001	0.001	0	0.2	0.2	0.04
Anfibio	0.55	0.0001	0.0006	0	0.28	0.118	0.0592
Aves	0.375	-0.0001	0.0003	-0	-0.1111	0.0625	-0.0204
Mamífero	0.2857	-0.0001	0.0002	-0	-0.1111	0.0426	-0.0204
Reptiles	0.2857	-0.0001	0.0002	-0	-0.1111	0.0426	-0.0204

Tabla I.4

	Esqueleto Respiración Nº Patas			Sí Pulmonar 4			
E4	C(r) C(c)	D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Vertebrado	1	0.0001	0.001	0	0.2	0.2	0.04
Mamífero	0.65	0.0001	0.0008	0	0.28	0.154	0.0592
Reptiles	0.65	0.0001	0.0008	0	0.28	0.154	0.0592
Cánido	0.6033	0.0002	0.0008	0	0.328	0.149	0.0717
Félido	0.6033	0.0002	0.0008	0	0.328	0.149	0.0717
Tortuga	0.6033	0.0002	0.0008	0	0.328	0.149	0.0717
Sapo	0.5567	0.0002	0.0007	0	0.328	0.1314	0.0717
Rana	0.5567	0.0002	0.0007	0	0.328	0.1314	0.0717
Anfibio	0.55	0.0001	0.0006	0	0.28	0.118	0.0592
Aves	-0.5	0.0001	-0.0001	0	0.1	-0.0278	0.02
Cetáceo	-0.8235	-0	-0.0004	-0	-0.0476	-0.0943	-0.0072
Serpiente	-0.8235	-0	-0.0004	-0	-0.0476	-0.0943	-0.0072

Tabla I.5

	Esqueleto Respiración Nº Patas Doméstico		Sí Pulmonar 4 Sí				
E5	C(r) C(c)	D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Vertebrado	1	0.0001	0.001	0	0.2	0.2	0.04
Perro	0.9426	0.0002	0.0014	0	0.3616	0.2595	0.081
Gato	0.7705	0.0002	0.0011	0	0.3616	0.2053	0.081
Tortuga_ Terrestre	0.6557	0.0002	0.0009	0	0.3616	0.1692	0.081
Mamífero	0.65	0.0001	0.0008	0	0.28	0.154	0.0592
Reptiles	0.65	0.0001	0.0008	0	0.28	0.154	0.0592
Cánido	0.6033	0.0002	0.0008	0	0.328	0.149	0.0717
Félido	0.6033	0.0002	0.0008	0	0.328	0.149	0.0717
Tortuga	0.6033	0.0002	0.0008	0	0.328	0.149	0.0717
Sapo	0.5567	0.0002	0.0007	0	0.328	0.1314	0.0717
Rana	0.5567	0.0002	0.0007	0	0.328	0.1314	0.0717
Anfibio	0.55	0.0001	0.0006	0	0.28	0.118	0.0592
Lobo	0.2828	0	0.0003	0	0.0025	0.0593	0.0028
León	0.2828	0	0.0003	0	0.0025	0.0593	0.0028
Aves	-0.5	0.0001	-0.0001	0	0.1	-0.0278	0.02
Cetáceo	-0.8235	-0	-0.0004	-0	-0.0476	-0.0943	-0.0072
Serpiente	-0.8235	-0	-0.0004	-0	-0.0476	-0.0943	-0.0072

Tabla I.6

	Esqueleto Respiración Nº Patas Doméstico Mama			Sí Pulmonar 4 Sí Sí			
E6	C(r) C(c)	D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Mamífero	1	0.0002	0.0018	0	0.424	0.3232	0.0968
Vertebrado	1	0.0001	0.001	0	0.2	0.2	0.04
Perro	0.9617	0.0002	0.0017	0	0.4042	0.3089	0.0933
Gato	0.847	0.0002	0.0014	0	0.4042	0.2583	0.0933
Cánido	0.8017	0.0002	0.0013	0	0.3952	0.2341	0.0903
Félido	0.8017	0.0002	0.0013	0	0.3952	0.2341	0.0903
Tortuga_ Terrestre	0.6557	0.0002	0.0009	0	0.3616	0.1692	0.081
Reptiles	0.65	0.0001	0.0008	0	0.28	0.154	0.0592
Tortuga	0.6033	0.0002	0.0008	0	0.328	0.149	0.0717
Sapo	0.5567	0.0002	0.0007	0	0.328	0.1314	0.0717
Rana	0.5567	0.0002	0.0007	0	0.328	0.1314	0.0717
Anfibio	0.55	0.0001	0.0006	0	0.28	0.118	0.0592
Lobo	0.5219	0	0.0006	0	0.069	0.122	0.0161
León	0.5219	0	0.0006	0	0.069	0.122	0.0161
Aves	-0.5	0.0001	-0.0001	0	0.1	-0.0278	0.02
Cetáceo	-0.6471	0	0.0001	0	0.055	0.0063	0.0129
Serpiente	-0.8235	-0	-0.0004	-0	-0.0476	-0.0943	-0.0072

Tabla I.7

	Esqueleto	Sí					
	Respiración	Pulmonar					
	Nº Patas	4					
	Doméstico	Sí					
	Mama	Sí					
	Raza	Siamés					
E7	C(r) C(c)	D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Mamífero	1	0.0002	0.0018	0	0.424	0.3232	0.0968
Vertebrado	1	0.0001	0.001	0	0.2	0.2	0.04
Gato	0.9847	0.0003	0.0023	0	0.5233	0.3918	0.1295
Cánido	0.8017	0.0002	0.0013	0	0.3952	0.2341	0.0903
Félido	0.8017	0.0002	0.0013	0	0.3952	0.2341	0.0903
Tortuga_ Terrestre	0.6557	0.0002	0.0009	0	0.3616	0.1692	0.0810
Reptiles	0.65	0.0001	0.0008	0	0.28	0.154	0.0592
Perro	0.6175	0.0001	0.0008	0	0.2552	0.1571	0.0555
Tortuga	0.6033	0.0002	0.0008	0	0.328	0.149	0.0717
Sapo	0.5567	0.0002	0.0007	0	0.328	0.1314	0.0717
Rana	0.5567	0.0002	0.0007	0	0.328	0.1314	0.0717
Anfibio	0.55	0.0001	0.0006	0	0.28	0.118	0.0592
Lobo	0.5219	0	0.0006	0	0.069	0.122	0.0161
León	0.5219	0	0.0006	0	0.069	0.122	0.0161
Aves	-0.5	0.0001	-0.0001	0	0.1	-0.0278	0.02
Cetáceo	-0.6471	0	-0.0001	0	0.055	0.0063	0.0129
Serpiente	-0.8235	-0	-0.0004	-0	-0.0476	-0.0943	-0.0072

Tabla I.8

	Pares_Nervios_Craneales 12 Número_Cámaras_Corazón 4 Nº Patas 4						
E8	C(r) C(c)	D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Mamífero	0.51	0.0002	0.0006	0	0.36	0.1164	0.0784
Reptiles	0.51	0.0002	0.0006	0	0.36	0.1164	0.0784
Cánido	0.4942	0.0002	0.0006	0	0.352	0.1156	0.078
Félido	0.4942	0.0002	0.0006	0	0.352	0.1156	0.078
Tortuga	0.4942	0.0002	0.0006	0	0.352	0.1156	0.078
Aves	-0.5918	0.0001	-0.0002	0	0.2	-0.0493	0.04
Cetáceo	-0.8616	-0	-0.0006	0	-0.0123	-0.1285	-0.0004
Serpiente	-0.8616	-0	-0.0006	0	-0.0123	-0.1285	-0.0004
Sapo	-1	-0.0001	-0.0017	-0	-0.2	-0.3191	-0.04
Rana	-1	-0.0001	-0.0017	-0	-0.2	-0.3191	-0.04
Anfibio	-1	-0.0002	-0.002	-0	-0.36	-0.36	-0.0784

Tabla I.9

	Pares_Nervios_Craneales 12 Número_Cámaras_Corazón 4 Nº Patas 0 Vive_En Tierra						
E9	C(r) C(c)	D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Cetáceo	0.9277	0.0002	0.0012	0	0.352	0.2285	0.078
Serpiente	0.9277	0.0002	0.0012	0	0.352	0.2285	0.078
Mamífero	0.51	0.0002	0.0006	0	0.36	0.1164	0.0784
Reptiles	0.51	0.0002	0.0006	0	0.36	0.1164	0.0784
Cánido	-0.0311	-0	-0	0	-0.0123	-0.001	-0.0004
Félido	-0.0311	-0	-0	0	-0.0123	-001	-0.0004
Tortuga	-0.0311	-0	-0	0	-0.0123	-001	-0.0004
Aves	0.5918	0.0001	-0.0002	0	0.2	-0.0493	0.04
Sapo	-1	-0.0003	-0.0023	-0	-0.488	-0.3984	-0.1153
Rana	-1	-0.0003	-0.0023	-0	-0.488	-0.3984	-0.1153
Anfibio	-1	-0.0002	-0.002	-0	-0.36	-0.36	-0.0784

Tabla I.10

	Nº Patas Color Respiración			4 Verde Pulmonar			
E10	C(r) C(c)	D1(r) C(r)	C(r) D1(c)	D1(r) D1(c)	D2(r) C(c)	C(r) D2(c)	D2(r) D2(c)
Tortuga_ Marina	0.7705	0.0002	0.0009	0	0.328	0.1825	0.0717
Sapo	0.468	0.0002	0.0005	0	0.424	0.1066	0.0968
Rana	0.468	0.0002	0.0005	0	0.424	0.1066	0.0968
Cánido	0.405	0.0001	0.0004	0	0.28	0.0882	0.0592
Félido	0.405	0.0001	0.0004	0	0.28	0.0882	0.0592
Tortuga	0.405	0.0001	0.0004	0	0.28	0.0882	0.0592
Mamífero	0.3	0.0001	0.0003	0	0.2	0.06	0.04
Reptiles	0.3	0.0001	0.0003	0	0.2	0.06	0.04
Anfibio	0.1	0.0001	0.0001	0	0.2	0.02	0.04
Tortuga_ Terrestre	-0.6078	-0	-0.0005	-0	-0.0476	-0.0953	-0.0072
Aves	-0.75	0	-0.0006	0	0	-0.125	0
Cetáceo	-0.8824	-0.0001	-0.0008	-0	-0.1111	-0.1546	-0.0204
Serpiente	-0.8824	-0.0001	-0.0008	-0	-0.1111	-0.1546	-0.0204

Tabla I.11

ANEXO II
PROGRAMAS COMO MARCOS

almacenando un cero en la ranura *Resultado* del marco instanciado *Operador_División_1.*

En este instante, solamente se conoce el valor que toma el primer operando del marco instanciado *Operador_Mayor_1*, pasándose entonces a ejecutar el segundo operando, es decir, la operación *(c*d)*. La ejecución de este operando se realiza enviando un mensaje a la ranura *Ejecutar* del marco instanciado *Operador_Multiplicación_1,* solicitándose así la ejecución del procedimiento:

Multiplicar (c, d, Resultado).

Como la ranura que ejecuta el mensaje no se encuentra en el marco instanciado y sí en su marco clase padre, y como los parámetros *c* y *d* son variables de Tipo Básico, se pasa a ejecutar la operación en la computadora. Con este fin, se envía un mensaje al objeto *c* de la *Jerarquía de Tipos* con el operando Multiplicación y con *d* como parámetro, ejecutándose la operación *a * b* físicamente en la computadora y almacenando el valor doce en la ranura *Resultado* del marco instanciado *Operador_Multiplicación_1.*

En este momento, el control lo posee el marco instanciado *Operador_Mayor_1* y, al conocerse que el primer operando del marco instanciado *Operador_Mayor_1* toma el valor cero y el segundo toma el valor doce, se pasa a ejecutar físicamente la operación:

Mayor (0,12, Resultado),

almacenando en *Resultado* el valor Falso y devolviendo el control al marco instanciado *Operador_Y_1.*

Al conocerse el valor que toma el primer operando del marco instanciado *Operador_Y_1*, se pasa a ejecutar el segundo operando de forma similar a las ejecuciones anteriores. La ejecución de la operación *(e=<f)*, con los valores anteriormente dados (e=5, f=6), hace al segundo operador cierto. Por tanto, se pasa a ejecutar físicamente la operación:

Y (Falso, Cierto, Resultado)

y se almacena en la ranura *Resultado* el valor Falso. en este momento, el control se devuelve al marco que hubiera originado la ejecución de la operación.

II. PROGRAMAS COMO MARCOS

II.1 INTRODUCCION

En este anexo se explican la jerarquía orientada a objetos y las tres jerarquías orientadas a marcos que van a permitir que el IC construya programas escritos en un lenguaje de alto nivel. Todas las jerarquías, ya sean de objetos o de marcos, han sido programadas en el lenguaje de orientación a objetos C++. No obstante, la descripción de las cuatro jerarquías se realiza independientemente del lenguaje en el que han sido programadas. Estas jerarquías, como se ve en la figura II.1, son y se relacionan de la siguiente forma:

1. *Jerarquía de Tipos:*

   Jerarquía de objetos que describe los tipos básicos más utilizados en un lenguaje de alto nivel y sus operaciones asociadas.

2. *Jerarquía de Operaciones:*

   Jerarquía de marcos clase que describe la mayoría de las operaciones que se pueden realizar sobre los tipos básicos de la *Jerarquía de Tipos*. Como cualquier lenguaje de alto nivel, la *Jerarquía de Operaciones* permite construir operaciones simples y operaciones compuestas.

3. *Jerarquía de Sentencias:*

   Jerarquía de marcos clase que describe el conjunto de sentencias secuenciales, alternativas, repetitivas y de entrada/salida más utilizadas en un lenguaje de alto nivel. La *Jerarquía de Sentencias* permite anidar sentencias y construir sentencias compuestas en las que pueden intervenir o no *Operaciones* y *Tipos Básicos* definidos en la *Jerarquía de Operaciones* y en la *Jerarquía de Tipos*.

4. *Jerarquía de Programas.*

   Representa un programa que está formado por un conjunto de *Sentencias* y *Tipos*.

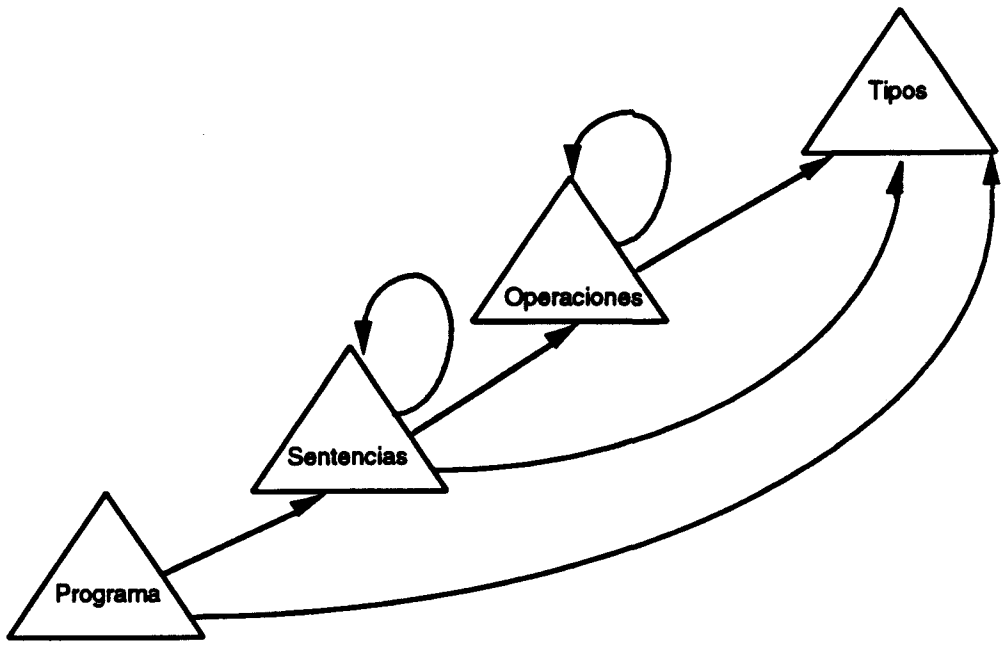


Figura II.1. Relaciones entre las jerarquías de *Tipos*, *Operaciones*, *Sentencias* y *Programas*.

Dado un programa P, cualquier sentencia u operación que en él aparezca será un marco instanciado de la sentencia u operador que represente. Por ejemplo, la sentencia de asignación *a:=b* es un marco instanciado del marco clase que representa la sentencia de asignación.

II.2 JERARQUIA DE TIPOS

La *Jerarquía de Tipos* es una jerarquía de objetos que clasifica los tipos de datos simples más utilizados en los lenguajes de programación de alto nivel. Los *Tipos_Compuestos*, o conjuntos de elementos del mismo tipo, se definen como un conjunto de *Tipos_Simples*. La figura II.2 muestra la *Jerarquía de Tipos* que se ha construido en este sistema.

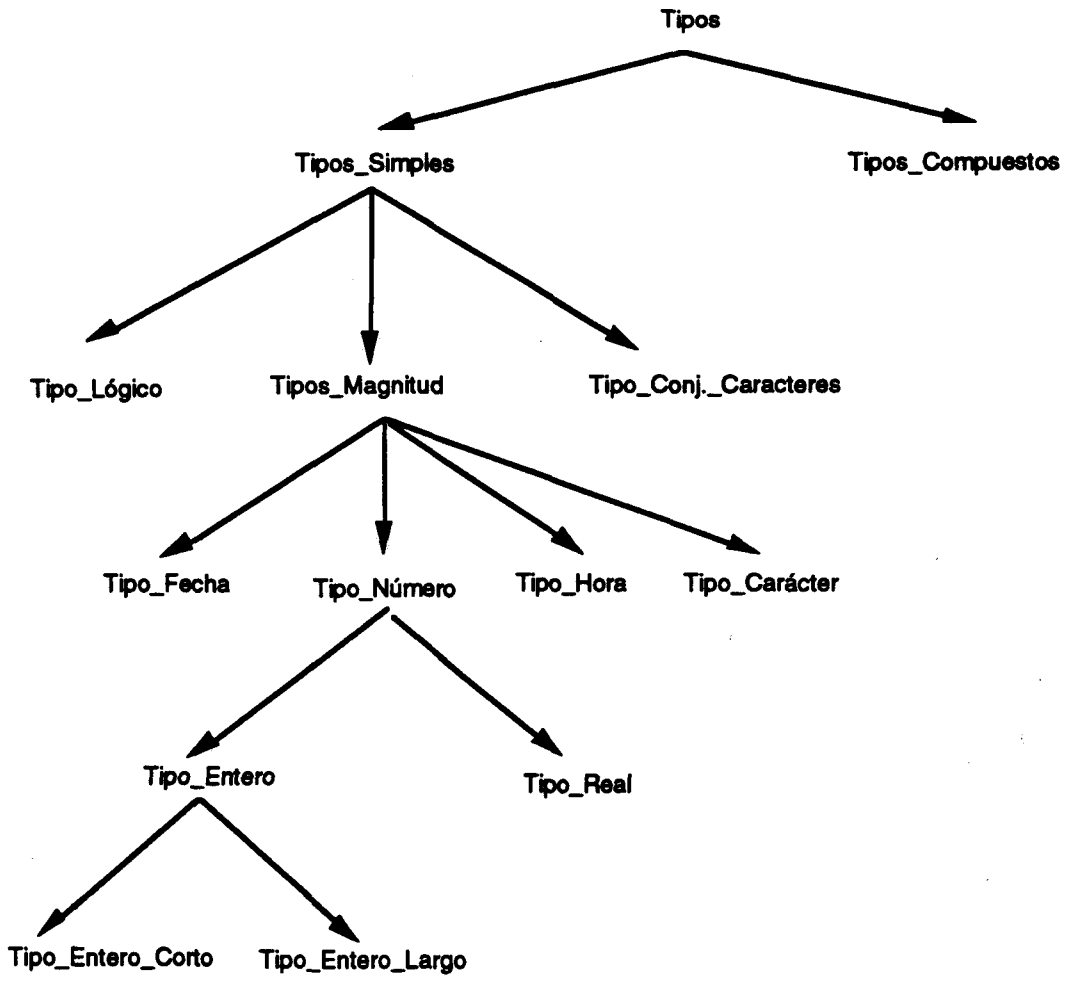


Figura II.2. Jerarquía de Tipos

Los métodos o procedimientos que, desde el punto de vista de la orientación a objetos, se han construido para cada clase terminal y no terminalde la jerarquía y su significado, se muestran en la tabla II.1. También se muestran en dicha tabla la estructura de cada clase.

CLASE	METODO	SIGNIFICADO
Tipo_Compuesto *Lista* *Numero_Elementos* *Tipo*		
Tipo_Simple		

Tabla II.1 Métodos definidos en la Jerarquía de Tipos (Continúa)

CLASE	METODO	SIGNIFICADO
Tipo_Lógico *Booleno*	Y (Operador1, Operardor2, Resultado)	Realiza la operación Y entre *Operador1* y *Operador2* y devuelve el *Resultado.*
	O (Operador1, Operardor2, Resultado)	Realiza la operación O entre *Operador1* y *Operador2* y devuelve el *Resultado.*
	NO (Operador1, Resultado)	La negación del *Operador1* se devuelve en *Resultado.*
Tipo_Magnitud	SUMA (Operador1, Operardor2, Resultado)	Realiza la SUMA del *Operador1* con el *Operador2* y devuelve el *Resultado*
	RESTA (Operador1, Operardor2, Resultado)	Realiza la RESTA del *Operador1* con el *Operador2* y devuelve el *Resultado*
	IGUAL (Operador1, Operardor2, Resultado)	Determina si el *Operador1* y *Operador2* son IGUALES y devuelve el *Resultado*
	DISTINTO(Operador1, Operardor2, Resultado)	Determina si el *Operador1* y *Operador2* son DISTINTOS y devuelve el *Resultado*
	MENOR (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es MENOR que el*Operador2* y devuelve el *Resultado*
	MAYOR (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es MAYOR que el *Operador2* y devuelve el *Resultado*
	MENOR_IGUAL (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es MENOR o IGUAL que el *Operador2* y devuelve el *Resultado*
	MAYOR_IGUAL (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es MAYOR o IGUAL que el *Operador2* y devuelve el *Resultado*
Tipo_Fecha *Dia* *Mes* *Año*		
Tipo_Hora *Horas* *Minutos* *Segundos*		
Tipo_Número	MULTIPLICACION (Operador1, Operardor2, Resultado)	Realiza la MULTIPLICACION del *Operador1* con el *Operador2* y devuelve el *Resultado*
	ABS (Operador1, Resultado)	Calcula el VALOR ABSOLUTO del *Operador1* y devuelve el *Resultado*
	NEG (Operador1, Resultado)	Calcula el NEGATIVO del *Operador1* y devuelve el *Resultado*

Tabla II.1 Métodos definidos en la Jerarquía de Tipos (Continúa)

CLASE	METODO	SIGNIFICADO
Tipo_Entero	MODULO (Operador1, Operardor2, Resultado)	Realiza la DIVISION en MODULO entre el *Operador1* y el *Operador2* y devuelve el *Resultado*
	ELEVADO_A (Operador1, Operardor2, Resultado)	ELEVA el *Operador1* al *Operador2* y devuelve el *Resultado*
	MANTISA (Operador1, Resultado)	Calcula la MANTISA del *Operador1* y devuelve el *Resultado*
	DIVISION ENTERA (Operador1, Operardor2, Resultado)	Realiza la DIVISION ENTERA del *Operador1* con el *Operador2* y devuelve el *Resultado*
Tipo_Entero_Corto *Entero_Corto*		
Tipo_Entero_Largo *Entero_Largo*		
Tipo_Real *Real*	DIVISION (Operador1, Operador2, Resultado)	Realiza la DIVISION del *Operador1* con el *Operador2* y devuelve el *Resultado*
	EXPONENTE (Operador1, Resultado)	Calcula el EXPONENTE del *Operador1* y devuelve el *Resultado*
	PARTE_ENTERA (Operador1, Resultado)	Calcula la PARTE ENTERA del *Operador1* y devuelve el *Resultado*
Tipo_Carácter *Letra*	CHR (Operador1, Resultado)	Calcula el CARACTER correspondiente al valor del *Operador1* y devuelve el *Resultado*
	ORD (Operador1, Resultado)	Calcula el valor ASCII correspondiente al valor del *Operador1* y devuelve el *Resultado*
Tipo_Conj._ Caracteres *String*	CONCATENACION (Operador1, Operardor2, Resultado)	Realiza la CONCATENACION del *Operador1* con el *Operador2* y devuelve el *Resultado*
	ASIGNACION (Operador1, Resultado)	ASIGNA el *Operador1* al *Resultado*
	MENOR (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es MENOR que el *Operador2* y devuelve el *Resultado*
	MAYOR (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es MAYOR que el *Operador2* y devuelve el *Resultado*
	DISTINTO (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es DISTINTO al *Operador2* y devuelve el *Resultado*
	IGUAL (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es IGUAL que el *Operador2* y devuelve el *Resultado*
	MENOR_IGUAL (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es MENOR o IGUAL que el *Operador2* y devuelve el *Resultado*
	MAYOR_IGUAL (Operador1, Operardor2, Resultado)	Determina si el *Operador1* es MAYOR o IGUAL que el *Operador2* y devuelve el *Resultado*

Tabla II.1. Métodos definidos en la Jerarquía de Tipos.

II.3. JERARQUIA DE OPERACIONES

Las operaciones que se pueden realizar sobre los *Tipos* definidos en el apartado anterior se clasifican en operaciones aritméticas y lógicas. En las figuras II.3, II.4 y II.5, se muestra desglosada una clasificación de las operaciones más utilizadas.

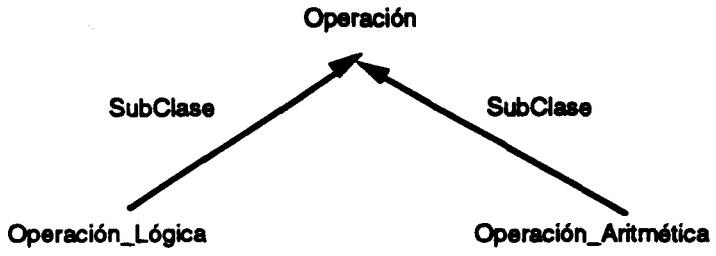


Figura II.3. Clasificación de las *Operaciones*

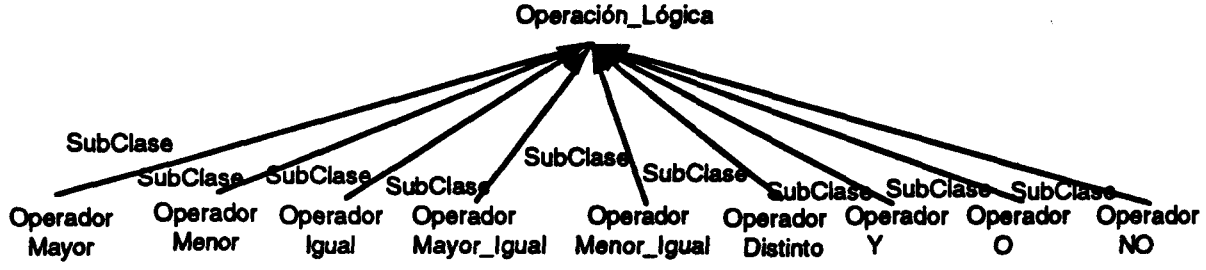


Figura II.4. Jerarquía de *Operaciones Lógicas*

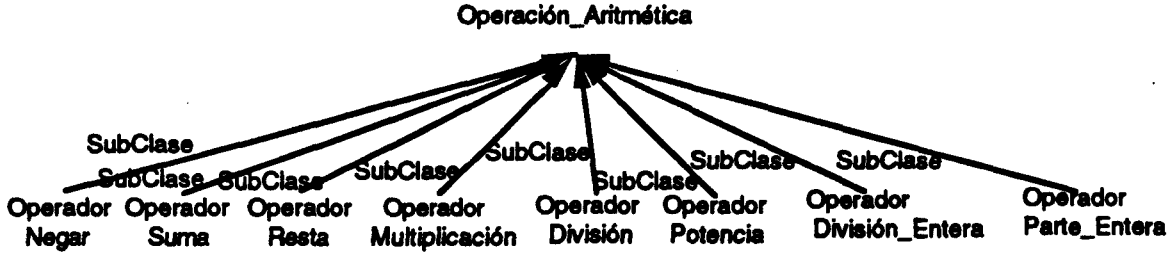


Figura II.5. Jerarquía de *Operaciones Aritméticas*

Todas las operaciones que aparecen en las jerarquías de la figura II.4 y II.5, salvo las operaciones *No, Negar y Parte_Entera,* son operaciones binarias en las que intervienen dos operandos, un operador y un resultado. El formato de cualquier operación será:

a) Si la operación es binaria: Resultado = Operando1 Operador Operando2

b) Si la operación es unaria: Resultado = Operador Operando1

Cada nodo o marco clase de las jerarquías II.4 y II.5, representa un tipo de operación y recibe el nombre del operador que interviene en la operación. Por ejemplo, la operación *Suma* se representa mediante el marco clase *Operador_Suma* en la *Jerarquía de Operaciones Aritméticas*. Si en un programa aparece la operación: *a+b*, el marco que representa esta operación es un marco instanciado del marco clase *Operador_Suma*.

Las operaciones aritméticas y lógicas se realizan sobre elementos básicos definidos en la *Jerarquía de Tipos* o sobre otras operaciones más sencillas definidas, de forma recursiva, en la *Jerarquía de Operaciones*, permitiéndo así la construcción de operaciones complejas. Con el fin de garantizar la compatibilidad de tipos de los operandos y la construcción de expresiones complejas correctas, en cada marco clase que representa una operación se especifica, para cada operando y resultado, sus tipos permitidos. Por tanto, para cada marco clase de la figura II.4 y II.5 se describirán sus operandos, su resultado y la ejecución de la operación. El orden de prelación de los operandos está incluido en el compilador que traduce cualquier operación a marcos instanciado.

En los siguientes apartados se describen las jerarquías de operaciones lógicas y aritméticas, la representación y ejecución de una operación compleja y el conjunto de marcos clase que permiten representar las operaciones de un lenguaje de alto nivel como marcos instanciados.

II.3.1 OPERACIONES LOGICAS

En este apartado se describen los marcos clase que forman la jerarquía de operaciones lógicas. El esquema que se ha seguido en la exposición de las operaciones lógicas es la descripción de los operandos y del resultado de la operación.

II.3.1.1 Descripción de los Operandos.

El tipo de valores que puede tener un operando en una operación lógica depende de la semántica de la operación. Básicamente se distinguen los siguientes tres casos:

* Operandos de tipo lógico

* Operandos de tipo conjunto de caracteres y de tipo magnitud

* Operandos de tipo simple

* Operandos de tipo operación

Se pasa a describir cada uno de ellos:

a) *Operandos de Tipo Lógico*

En los marcos *Operador_No*, *Operador_Y* y *Operador_O*, el(los) operando(s) se definen de *Tipo_Lógico*, o, recursivamente, como una *Operación_Lógica*, permitiéndose así la definición de expresiones lógicas compuestas.

b) *Operandos de Tipo Conjunto_Caracteres y de Tipo Magnitud*

Para los marcos *Operador_Mayor*, *Operador_Menor*, *Operador_Mayor_igual* y *Operador_Menor_Igual*, se definen los operandos: *Operando1* y *Operando2* de *Tipo_Conjunto_Caracteres* y de *Tipo_Magnitud*. Se define de *Tipo_Magnitud* y no de *Tipo_Carácter*, *Tipo_Numero*, *Tipo_Fecha* y *Tipo_Hora*, porque todas las operaciones son válidas en los *Tipos* anteriormente mencionados.

c) *Operandos de Tipo Simple*

En los marcos *Operador_Igual* y *Operador_Distinto* los operandos se definen de *Tipo_Simple* porque estas operaciones son válidas para los tipos: *Tipo_Lógico*, *Tipo_Magnitud* y *Tipo_Conj._Caracteres*.

d) *Operandos de Tipo Operación*

Además, para permitir el uso de expresiones lógicas complejas que incluyan la evaluación de operaciones aritméticas, es necesario que los operandos lógicos se definan como *Operaciones* y no como *Operaciones_Lógicas*.

II.3.1.2 Resultado de la Operación.

La evaluación de una operación lógica origina siempre un *Resultado* de *Tipo_Lógico*, motivo por el cual se ha definido la propiedad *Resultado* en el Marco *Operación_Lógica* y no en los Marcos del nivel inmediatamente inferior que representan cada uno de los tipos de operaciones lógicas.

II.3.2 OPERACIONES ARITMETICAS.

Similar a las *Operaciones_Lógicas* se comportan las *Operaciones_Aritméticas*. Se pasa a describir el conjunto de valores que toman los operandos y el resultado en estas operaciones.

II.3.2.1 Descripción de los Operandos.

El tipo de valores que puede tener un operando en una operación aritmética depende de la semántica de la operación y son:

a) Los marcos *Operador_Negar*, *Operador_Potencia* y *Operador_Multiplicación*, al poder realizar las operaciones sobre un *Tipo_Entero* o sobre un *Tipo_Real*, sus operandos se han definido de *Tipo_Numero*. No ocurre así con el *Operador_División* y con el *Operador_Parte_Entera* en los que los operandos son de *Tipo_Real*.

b) El operador más restringido es el operador representado en el marco *Operador_División_Entera*, operador que necesita como operandos elementos de *Tipo_Entero*.

c) En el punto opuesto se encuentran el *Operador_Suma* y el *Operador_Resta*, operadores que, además de sumar y restar números enteros y reales, pueden sumar o restar horas, fechas, caracteres e incluso cadenas de caracteres, entendiendo este último caso como la concatenación y el borrado de palabras. Por estos motivos, los operandos de los operadores suma y resta, además de ser de *Tipo_Conj._Caracteres*, pueden ser de *Tipo_Numero*, *Tipo_Fecha*, *Tipo_Hora* y *Tipo_Carácter*, es decir, de *Tipo_Magnitud*.

II.3.4 OPERACIONES COMO MARCOS

En esta sección, cada uno de los marcos que aparecen en las figuras II.3, II.4 y II.5, se han representado en forma de tabla. Así, las tablas número II.2, II.3 y II.13, representan, respectivamente, las operaciones en general, las operaciones lógicas y las operaciones aritméticas. Las tablas comprendidas entre la II.4 y la II.12, cada una de las operaciones lógicas y, por último, las tablas comprendidas entre la II.14 y la II.21 las operaciones aritméticas.

Operaciones	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase		
SuperClase	Relación_SuperClase	Operación_Lógica Operación_Aritmética	
Nombre_Marco	Tipo_Conj._Caracteres	Operaciones	
(*) Nombre_Operación	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres

Tabla II.2. Marco Operaciones

Operación_Lógica	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operaciones	
SuperClase	Relación_SuperClase	Operador_Y Operador_O Operador_No Operador_Mayor Operador_Menor Operador_Igual Operador_Mayor_Igual Operador_Menor_Igual Operador_Distinto	
Nombre_Marco	Tipo_Conj._Caracteres	Operación_Lógica	
(*) Resultado	Tipo_Lógico		Tipo_Lógico

Tabla II.3. Marco Operación_Lógica

Operador_Y	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Lógica	
Símbolo	Tipo_Carácter	Y, ∧	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Y	
Ejecutar	Método	Y (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Lógico Marco Operación_Lógica		Tipo_Lógico Puntero
(*) *Operando2*	Tipo_Lógico Marco Operación_Lógica		Tipo_Lógico Puntero

Tabla II.4. Marco Operador_Y

Operador_O	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Lógica	
Símbolo	Tipo_Carácter	O, ∨	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_O	
Ejecutar	Método	O (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Lógico Marco Operación_Lógica		Tipo_Lógico Puntero
(*) *Operando2*	Tipo_Lógico Marco Operación_Lógica		Tipo_Lógico Puntero

Tabla II.5. Marco Operador_O

Operador_No	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Lógica	
Símbolo	Tipo_Conj._Caracteres	NO, ¬	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_No	
Ejecutar	Método	No (Operando1, Resultado)	
(*) *Operando1*	Tipo_Lógico Marco Operación_Lógica		Tipo_Lógico Puntero

Tabla II.6. Marco Operador_NO

Operador_Mayor	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Lógica	
Símbolo	Tipo_Carácter	>	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Mayor	
Ejecutar	Método	Mayor (Operando1, Operando2, Resultado)	
(*) Operando1	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operaciones		Tipo_Magnitud Tipo_Conj._Caracteres Puntero
(*) Operando2	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operaciones		Tipo_Magnitud Tipo_Conj._Caracteres Puntero

Tabla II.7. Marco Operador_Mayor

Operador_Menor	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Lógica	
Símbolo	Tipo_Carácter	<	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Menor	
Ejecutar	Método	Menor (Operando1, Operando2, Resultado)	
(*) Operando1	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operaciones		Tipo_Magnitud Tipo_Conj._Caracteres Puntero
(*) Operando2	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operaciones		Tipo_Magnitud Tipo_Conj._Caracteres Puntero

Tabla II.8. Marco Operador_Menor

Operador_Igual	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Lógica	
Símbolo	Tipo_Carácter	=	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Igual	
Ejecutar	Método	Igual (Operando1, Operando2, Resultado)	
(*) Operando1	Tipo_Simple Marco Operaciones		Tipo_Simple Puntero
(*) Operando2	Tipo_Simple Marco Operaciones		Tipo_Simple Puntero

Tabla II.9. Marco Operador_Igual

Operador_ Mayor_Igual	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Lógica	
Símbolo	Tipo_Conj._Caracteres	>=	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Mayor_Igual	
Ejecutar	Método	Mayor_Igual (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operaciones		Tipo_Magnitud Tipo_Conj._Caracteres Puntero
(*) *Operando2*	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operaciones		Tipo_Magnitud Tipo_Conj._Caracteres Puntero

Tabla II.10. Marco Operador_Mayor_Igual

Operador_ Menor_Igual	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Lógica	
Símbolo	Tipo_Conj._Caracteres	<=	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Menor_Igual	
Ejecutar	Método	Menor_Igual (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operaciones		Tipo_Magnitud Tipo_Conj._Caracteres Puntero
(*) *Operando2*	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operaciones		Tipo_Magnitud Tipo_Conj._Caracteres Puntero

Tabla II.11. Marco Operador_Menor_Igual

Operador_Distinto	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Lógica	
Símbolo	Tipo_Carácter	≠	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Distinto	
Ejecutar	Método	Distinto (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Simple Marco Operaciones		Tipo_Simple Puntero
(*) *Operando2*	Tipo_Simple Marco Operaciones		Tipo_Simple Puntero

Tabla II.12. Marco Operador_Distinto

Operación_Aritmética	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Operaciones
SuperClase	Relación_SuperClase	Operador_Negar Operador_Suma Operador_Resta Operador_Multiplicación Operador_División Operador_Potencia Operador_División_Entera Operador_Parte_Entera
Nombre_Marco	Tipo_Conj._Caracteres	Operación_Aritmética

Tabla II.13. Marco Operación_Aritmética

Operador_Suma	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Aritmética	
Símbolo	Tipo_Carácter	+	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Suma	
Ejecutar	Método	Suma (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operación_Aritmética		Tipo_Magnitud Tipo_Conj._Caracteres Puntero
(*) *Operando2*	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operación_Aritmética		Tipo_Magnitud Tipo_Conj._Caracteres Puntero
(*) *Resultado*	Tipo_Magnitud Tipo_Conj._Caracteres		Tipo_Magnitud Tipo_Conj._Caracteres

Tabla II.14. Marco Operador_Suma

Operador_Resta	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Aritmética	
Símbolo	Tipo_Carácter	-	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Resta	
Ejecutar	Método	Resta (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operación_Aritmética		Tipo_Magnitud Tipo_Conj._Caracteres Puntero
(*) *Operando2*	Tipo_Magnitud Tipo_Conj._Caracteres Marco Operación_Aritmética		Tipo_Magnitud Tipo_Conj._Caracteres Puntero
(*) *Resultado*	Tipo_Magnitud Tipo_Conj._Caracteres		Tipo_Magnitud Tipo_Conj._Caracteres

Tabla II.15. Marco Operador_Resta

Operador_ Multiplicación	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Aritmética	
Símbolo	Tipo_Carácter	*	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Multiplicación	
Ejecutar	Método	Multiplicar (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Número Marco Operación_Aritmética		Tipo_Número Puntero
(*) *Operando2*	Tipo_Número Marco Operación_Aritmética		Tipo_Número Puntero
(*) *Resultado*	Tipo_Número		Tipo_Número

Tabla II.16. Marco Operador_Multiplicación

Operador_División	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Aritmética	
Símbolo	Tipo_Carácter	/	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_División	
Ejecutar	Método	Dividir (Operador1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Real Marco Operación_Aritmética		Tipo_Real Puntero
(*) *Operando2*	Tipo_Real Marco Operación_Aritmética		Tipo_Real Puntero
(*) *Resultado*	Tipo_Real		Tipo_Real

Tabla II.17. Marco Operador_División

Operador_Potencia	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Aritmética	
Símbolo	Tipo_Conj._Caracteres	**	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Potencia	
Ejecutar	Método	Potencia (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Número Marco Operación_Aritmética		Tipo_Número Puntero
(*) *Operando2*	Tipo_Número Marco Operación_Aritmética		Tipo_Número Puntero
(*) *Resultado*	Tipo_Número		Tipo_Número

Tabla II.18. Marco Operador_Potencia

Operador_ División_Entera	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Aritmética	
Símbolo	Tipo_Conj._Caracteres	Div	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_División_Entera	
Ejecutar	Método	División_Entera (Operando1, Operando2, Resultado)	
(*) *Operando1*	Tipo_Entero Marco Operación_Aritmética		Tipo_Entero Puntero
(*) *Operando2*	Tipo_Entero Marco Operación_Aritmética		Tipo_Entero Puntero
(*) *Resultado*	Tipo_Entero		Tipo_Entero

Tabla II.19. Marco Operador_División_Entera

Operador_ Parte_Entera	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Aritmética	
Símbolo	Tipo_Conj._Caracteres	Ent	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Parte_Entera	
Ejecutar	Método	Parte_Entera (Operando1, Resultado)	
(*) *Operando1*	Tipo_Real Marco Operación_Aritmética		Tipo_Real Puntero
(*) *Resultado*	Tipo_Real		Tipo_Real

Tabla II.20. Marco Operador_Parte_Entera

Operador_Negar	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Operación_Aritmética	
Símbolo	Tipo_Conj._Caracteres	Neg	
Nombre_Marco	Tipo_Conj._Caracteres	Operador_Negar	
Ejecutar	Método	Negar (Operando1, Resultado)	
(*) *Operando1*	Tipo_Número Marco Operación_Aritmética		Tipo_Número Puntero
(*) *Resultado*	Tipo_Número		Tipo_Número

Tabla II.21. Marco Operador_Negar

II.4 JERARQUIAS DE SENTENCIAS

La *jerarquía de marcos de la figura II.7, muestra una clasificación* de las sentencias secuenciales, alternativas, repetitivas y de entrada/salida más utilizadas en los lenguajes de alto nivel. Cualquier sentencia que aparezca en un programa será instancia de alguna de estas sentencias. Se pasa a describir cómo las distintas sentencias se representan como marcos.

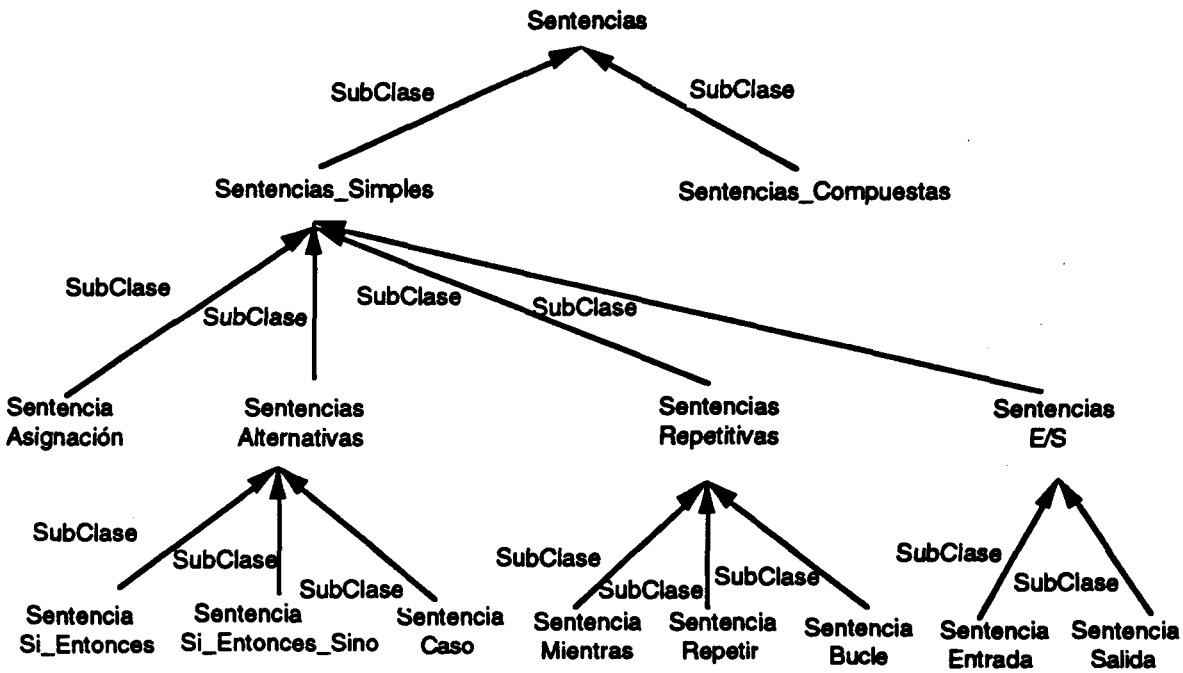


Figura II.7. Jerarquía de Sentencias

II.4.1 SENTENCIA DE ASIGNACION

Una sentencia de asignación es una *Sentencia_Simple* que, normalmente, se representa como:

$$\langle P^{er}\_Miembro \rangle := \langle S^{o}\_Miembro \rangle ,$$

donde $P^{er}\_Miembro$ es una variable a la que se le asignan valores de variables u operaciones definidas en el $S^{o}\_Miembro$. En cualquier caso, siempre debe existir una compatibilidad de tipos entre el primer y segundo miembro. La tabla II.24, representa el marco que describe la sentencia de asignación.

II.4.2 SENTENCIAS ALTERNATIVAS

Son tres las sentencias alternativas básicas: *Sentencia_Si_Entonces*, *Sentencia_Si_Entonces_Sino* y *Sentencia_Caso* que se emplean en los lenguajes de alto nivel, y dos los elementos básicos que forman parte de cada una de ellas: condiciones y acciones. En cualquier sentecia alternativa existe una condición que debe ser evaluada y, en función de si se cumplen o no las condiciones, se pasa a ejecutar unas acciones.

Estas tres sentencias se representan, normalmente, en los lenguajes de alto nivel como:

Sentencia_Si_Entonces:	Si *<Condición>* Entonces *<Acción_Entonces>*
Sentencia_Si_Entonces_Sino:	Si *<Condición>* Entonces *<Acción_Entonces>* Sino *<Acción_Sino>*
Sentencia_Caso:	Caso *<Condición >* Hacer *<Caso_1>*: *<Acción_1>* ..... *<Caso_N>* : *<Acción_N>* En otro caso: *<Acción_Omisión>*

II.4.2.1 Condición

Pueden darse en las *Sentencias_Alternativas* dos situaciones:

a) Si la sentencia que se está representando es una *Sentencia_Si_Entonces* o una *Sentencia_Si_Entonces_Sino*, la *Condición* será siempre una *Operación_Lógica* o de *Tipo_Lógico*.

b) Si la sentencia es una *Sentencia_Caso*, la condición podrá ser, además de una *Operación_Lógica*, una *Operación_Aritmética*, motivo por el cual se define la condición en esta sentencia de tipo *Operación*. Ademas, al poder preguntar en la *Condición* de la *Sentencia_Caso* por valores de *Tipo_Fecha*, *Tipo_Carácter* y *Tipo_Número*, entre otros, se definirá también de *Tipo_Simple*.

II.4.2.2 Acción

Las acciones asociadas a las sentencias alternativas son *Sentencias_Compuestas* formadas por un conjunto de *Sentencias_Simples* o de *Sentecias_Compuestas*:

a) Si se tiene una *Sentencia_Si_Entonces* y la *Condición* es cierta, se pasa a *Ejecutar* la *Acción_Entonces*.

b) Si se tiene una *Sentencia_Si_Entonces_Sino* y la *Condición* es cierta, se pasa a *Ejecutar* la *Acción_Entonces* y, si es falsa, se ejecuta la *Acción_Sino*.

c) Si se tiene una *Sentencia_Caso* se pasa a ejecutar la *Acción_i* asociada al *Caso_i* que es cierto. En caso contrario, se ejecuta una *Acción_por_Omisión*.

II.4.2.3 Ejecución

Para ejecutar cualquier *Sentencia_Si_Entonces*, *Sentencia_Si_Entonces_Sino* y *Sentencia_Caso* que aparezca en un programa, se envia un mesaje solicitando la ejecución de la sentencia implicada, sentencia representada como un marco instanciado del marco clase *Sentencia_Si_Entonces*, *Sentencia_Si_Entonces_Sino* y *Sentencia_Caso* respectivamente.

Al no encontrarse la ranura *Ejecutar* en el marco instanciado, el SBM, aplicando herencia de propiedades, accede al marco clase y, al encontrarse la ranura buscada, ejecuta en cada caso el procedimiento:

Sentencia_Si_Entonces (Condición, Acción_Entonces) o,

Sentencia_Si_Entonces_Sino (Condición, Acción_Entonces, Acción_Sino) o,

Sentencia_Caso (Condición, Nº_Casos, Caso1, Acción_1, ..., Caso_n, Acción_n, Acción_por_Omisión),

procedimientos que envían un mensaje al marco instanciado que representa la *Condición* de la sentencia implicada. Evaluada la condición se pasa a ejecutar, en cada caso, la acción correspondiente según lo descrito en el apartado anterior.

II.4.3 SENTENCIAS REPETITIVAS

Son tres las sentencias repetitivas básicas: *Sentencia_Mientras*, *Sentencia_Repetir* y *Sentencia_Bucle* que se emplean en los lenguajes de alto nivel, y, al igual que ocurría en las sentencias alternativas, los dos elementos básicos que forman parte de cada una de ellas son: condiciones y acciones. En cualquier sentencia repetitiva existe un conjunto de acciones que se ejecutan mientras se cumplan o no unas condiciones.

Estas tres sentencias se suelen representar en los lenguajes de alto nivel según el esquema siguiente:

Sentencia_Mientras:	Mientras *<Condición>* Hacer *<Acción>*
Sentencia_Repetir:	Repetir *<Acción>* Hasta *<Condición>*
Sentencia_Bucle:	Desde *<Desde>* Hasta *<Hasta>*: Incremento *<Incremento>* Hacer: *<Acción>*

II.4.3.1 Condición

En todas las *Sentencias_Repetitivas*, la *Condición* es siempre una operación lógica. En el caso de la *Sentencia_Bucle*, la condición equivale a comprobar la desigualdad: *Desde* < *Variable* < *Hasta*.

II.4.3.2 Acciones

Las acciones asociadas a las sentencias repetitivas son *Sentencias_Compuestas* formada por un conjunto de *Sentencias_Simples* o de *Sentencias_Compuestas*:

a) Si se tiene una *Sentencia_Mientras*, la *Acción* se está ejecutando mientras la *Condición* es cierta. Cuando la *Condición* es falsa, la *Acción* deja de ejecutarse.

b) Si se tiene una *Sentencia_Repetir*, la *Acción* se está ejecutando mientras la *Condición* es falsa. Cuando la *Condición* es cierta, la *Acción* deja de ejecutarse.

c) Si se tiene una *Sentencia_Bucle* la *Acción* se está ejecutando mientras no se alcance el límite superior, sumándose en cada paso de la ejecución el valor del incremento al contador.

II.4.3.2 Ejecución

Para ejecutar cualquier *Sentencia_Mientras*, *Sentencia_Repetir* y *Sentencia_Bucle* que aparezca en un programa, es necesario enviar un mesaje a la ranura *Ejecutar* de la sentencia implicada, sentencia representada como un marco instanciado del marco clase *Sentencia_Mientras*, *Sentencia_Repetir* y *Sentencia_Bucle* respectivamente. Aplicando herencia de propiedades, se accederá al marco clase y, en cada caso, se ejecutará el procedimiento:

Sentencia_Mientras (Condición, Acción) o,

Sentencia_Repetir (Acción, Condición) o,

Sentencia_Bucle (Desde, Hasta, Incremento, Acción),

procedimientos que simulan la ejecución de las sentencias tal cual ocurren en un lenguaje de alto nivel.

Evaluada la condición se pasa a ejecutar, en cada caso, la acción correspondiente según lo descrito en el apartado anterior.

II.4.4 SENTENCIAS DE ENTRADA/SALIDA

Las sentencias de entrada/salida son sentencias que imprimen una *cadena* de caracteres y obtienen/imprimen los valores de unos *argumentos*. Se expresan como:

Sentencia_Salida: Visualizar (*Cadena, Argumento1, ArgumentoN*)

Sentencia_Entrada: Entrada (*Cadena, Argumento1, ArgumentoN*)

II.4.5 SENTENCIAS COMO MARCOS

En esta sección, cada uno de los marcos clase que aparecen en la Jerarquía de la figura II.7, se han representado en forma de tabla. Así, las tablas número II.22 y II.23, representan, respectivamente, las sentencias simples y compuestas; la tabla II.24, la sentencia de asignación; las tablas comprendidas entre la II.25 y la II.28, ambas inclusive, las sentencias alternativas; las comprendidas entre las II.29 y II.32 las sentencias repetitivas; y, por último, las tablas comprendidas entre la II.33 y la II.35 las sentencias de entrada/salida.

Sentencias_Simples	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias	
SuperClase	Relación_SuperClase	Sentencia_Asignación Sentencias_Alternativas Sentencias_Repetitivas Sentencias_E/S	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencias_Simples	
(*) *Nombre_Sentencia*	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres

Tabla II.22. Marco *Sentencias_Simples*

Sentencias_Compuestas	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias	
Símbolo	Tipo_Conj._Caracteres	Comienzo <Sentencia1> <SentenciaN> Fin	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencias_Compuestas	
Ejecutar	Método	∀ Sentencia Simple *Ejecutar*	
(*) *Sentencia1*	Marco Sentencias_Simples Marco Sentencias_Compuestas		Puntero
(*) *SentenciaN*	Marco Sentencias_Simples Marco Sentencias_Compuestas		Puntero

Tabla II.23. Marco *Sentencias_Compuestas*

Sentencia_Asignación	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencia_Simple	
Símbolo	Tipo_Conj._Caracteres	:=	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencia_Asignación	
Ejecutar	Método	Sentencia_Asignación (P^{er}_Miembro, S^{o}_Miembro)	
(*) *P^{er}_Miembro*	Tipo_Simple		Tipo_Simple
(*) *S^{o}_Miembro*	Tipo_Simple Marco Operación		Tipo_Simple Puntero

Tabla II.24. Marco *Sentencia_Asignación*

Sentencias_Alternativas	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Sentencias_Simples
SuperClase	Relación_SuperClase	Sentencia_Si_Entonces Sentencia_Si_Entonces_Sino Sentencia_Caso
Nombre_Marco	Tipo_Conj._Caracteres	Sentencias_Alternativas

Tabla II.25. Marco *Sentencias_Alternativas*

Sentencia_Si_ Entonces	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias_Alternativa	
Símbolo	Tipo_Conj._Caracteres	Si <Condición> Entonces <Acción_Entonces>	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencia_Si_Entonces	
Ejecutar	Método	Sentencia_Si_Entonces (Condición, Acción_Entonces)	
(*) *Condición*	Tipo_Lógico Marco Operación_Lógica		Tipo_Lógico Puntero
(*) *Acción_Entonces*	Marco Sentencias_Compuestas		Puntero

Tabla II.26. Marco *Sentencia_Si_Entonces*

Sentencia_Si_Entonces_Sino	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias_Alternativa	
Símbolo	Tipo_Conj._Caracteres	Si <Condición> Entonces <Acción_Entonces> Sino <Acción_Sino>	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencia_Si_Entonces_Sino	
Ejecutar	Método	Sentencia_Si_Entonces_Sino (Condición, Acción_Entonces, Acción_Sino)	
(*) Condición	Tipo_Lógico Marco Operación_Lógica		Tipo_Lógico Puntero
(*) Acción_Entonces	Marco Sentencias_Compuestas		Puntero
(*) Acción_Sino	Marco Sentencias_Compuestas		Puntero

Tabla II.27. Marco *Sentencia_Si_Entonces_Sino*

Sentencia_Caso	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias_Alternativa	
Símbolo	Tipo_Conj._Caracteres	Caso <Condición> Hacer <Caso_1>: <Acción_1> <Caso_n>: <Acción_n> En otro caso: <Acción_Omisión>	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencia_Caso	
Ejecutar	Método	Sentencia_Caso (Condición, Nº_Casos, Caso1, Acción_1, ..., Caso_n, Acción_n, Acción_Omisión).	
(*) Nº_Casos	Tipo_Entero_Corto		Tipo_Entero_Corto
(*) Condición	Tipo_Simple Marco Operaciones		Tipo_Simple Puntero
(*) Caso_1	Tipo_Simple		Tipo_Simple
(*) Acción_1	Marco Sentencias_Compuestas		Puntero
(*) Caso_n	Tipo_Simple		Tipo_Simple
(*) Acción_n	Marco Sentencias_Compuestas		Puntero
(*) Acción_Omisión	Marco Sentencias_Compuestas		Puntero

Tabla II.28. Marco *Sentencia_Caso*

Sentencias_Repetitivas	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias_Simples	
SuperClase	Relación_SuperClase	Sentencia_Mientras Sentencia_Repetir Sentencia_Bucle	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencias_Repetitivas	
(*) Acción	Marco Sentencias_Compuestas		Puntero

Tabla II.29. Marco *Sentencias_Repetitivas*

Sentencia_Mientras	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias_Repetitivas	
Símbolo	Tipo_Conj._Caracteres	Mientras < Condición > Hacer < Acción >	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencia_Mientras	
Ejecutar	Método	Sentencia_Mientras (Condición, Acción)	
(*) Condición	Tipo_Lógico Marco Operación_Lógica		Tipo_Lógico Puntero

Tabla II.30 Marco *Sentencia_Mientras*

Sentencia_Repetir	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias_Repetitivas	
Símbolo	Tipo_Conj._Caracteres	Repetir < Acción > Hasta < Condición >	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencia_Repetir	
Ejecutar	Método	Sentencia_Repetir (Acción, Condición)	
(*) Condición	Tipo_Lógico Marco Operación_Lógica		Tipo_Lógico Puntero

Tabla II.31. Marco *Sentencia_Repetir*

Sentencia_Bucle	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias_Repetitivas	
Símbolo	Tipo_Conj._Caracteres	Desde < Desde > Hasta < Hasta > Incremento < Icremento > Hcer < Acción >	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencia_Bucle	
Ejecutar	Método	Sentencia_Bucle (Desde, Hasta, Incremento, Acción)	
(*) Desde	Tipo_Número		Tipo_Número
(*) Hasta	Tipo_Número		Tipo_Número
(*) Incremento	Tipo_Número		Tipo_Número

Tabla II.32. Marco *Sentencia_Bucle*

Sentencias_E/S	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
Clase	Relación_SubClase	Sentencias_Simples	
SubClase	Relación_SuperClase	Sentencia_Salida Sentencia_Entrada	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencias_E/S	
(*) Cadena	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*) Numero_Argumentos	Tipo_Entero_Corto		Tipo_Entero_Corto

Tabla II.33. Marco *Sentencias_E/S*

Sentencia_Salida	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias_E/S	
Símbolo	Tipo_Conj._Caracteres	Visualizar	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencia_Salida	
Ejecutar	Método	Visualizar (Cadena, Argumento1, ..., ArgumentoN)	
(*) Argumento1	Tipos_Simples Marco Operaciones		Tipos_Simples Puntero
(*) ArgumentoN	Tipos_Simples Marco Operaciones		Tipos_Simples Puntero

Tabla II.34. Marco *Sentencia_Salida*

Sentencia_Entrada	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Sentencias_E/S	
Símbolo	Tipo_Conj._Caracteres	Entrada	
Nombre_Marco	Tipo_Conj._Caracteres	Sentencia_Entrada	
Ejecutar	Método	Entrada (Cadena, Argumento1, ArgumentoN)	
(*) *Argumento1*	Tipos_Simples		Tipos_Simples
(*) *ArgumentoN*	Tipos_Simples		Tipos_Simples

Tabla II.35. Marco *Sentencia_Entrada*

II.5 JERARQUIA DE PROGRAMAS

El marco clase *Programa* contiene información sobre las *Precondiciones*, que deben cumplirse antes de *Ejecutar* las *Acciones* del programa. Ejecutar el programa con las *Precondiciones* satisfechas garantiza el cumplimiento de las *Postcondiciones* en él especificadas.

Programa	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
(*) *Nombre_Programa*	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*) *Precondición*	Tipo_Lógico Marco Operación_Lógica Marco Sentencias_Alternativas		Tipo_Lógico Puntero
(*) *Acción*	Marco Sentencias_Compuestas		Puntero
(*) *Postcondición*	Tipo_Lógico Marco Operación_Lógica Marco Sentencias_Alternativas		Tipo_Lógico Puntero
Ejecutar	Método	Ejecutar_Programa (Precondición, Acción, Postcondición)	

Tabla II.36. Marco *Programa*

ANEXO III.
MARCOS COMO MARCOS

III. MARCOS COMO MARCOS

III.1. INTRODUCCION

En este anexo, se muestran las tablas que formalizan el conocimiento declarativo de los marcos utilizando marcos y los métodos que se han utilizado en cada concepto.

Las tablas comprendidas entre la tabla III.1 y la III.3, representan los Marcos; la tabla III.4 el concepto de ranura; las tablas comprendidas entre la III.5 y la III.19, las relaciones, y, por último, las comprendidas entre la III.20 y la III.22, las propiedades. Posteriormente, se muestran los procedimientos asociados a cada una de los métodos que aparecen en dichas tablas.

III.1 TABLAS

Marco	TIPO RANURA	VALORES PERMITIDOS
(*) Identificador	Tipo_Conj._Caracteres	Tipo_Conj._Caracteres

Tabla III.1. Marco Marco

Marco_Instanciado	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
Nombre_Marco	Tipo_Conj._Caracteres	Marco__Instanciado	
SI añado	Método	Crear_Marco_Instanciado (Marco)	
SI borro	Método	Borrar_Marco_Instanciado (Marco)	
(*) Instancia	Marco Relación_Instancia		Puntero

Tabla III.2. Marco Marco_Instanciado

Marco_Clase	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
Nombre_Marco	Tipo_Conj._Caracteres	Marco_Clase	
SI Añado	Método	Crear_Marco_Clase (Marco)	
SI Borro	Método	Borrar_Marco_Clase (Marco)	
(*) SubClase	Marco Relación_SubClase		Puntero
(*) SuperClase	Marco Relación_SuperClase		Puntero
(*) Elementos_Clase	Marco Relación_Elementos_Clase		Puntero
(*) Fraternal	Marco Relación_Fraternal		Puntero
(*) Disjunto	Marco Relación_Disjunto		Puntero
(*) No Disjunto	Marco Relación_No_Disjunto		Puntero
(*) Pertenece	Marco Relación_Pertenece		Puntero
(*) Elementos_Conjunto	Marco Relación_Elementos_Conjunto		Puntero
(*) Ad_hoc	Marco Relación_Ad_Hoc		Puntero
(*) Propiedades	Marco Propiedad_de_Clase Marco Propiedad_de_Instancia Marco Marco_Clase		Puntero

Tabla III.3. Marco Marco_Clase

Ranuras	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SuperClase	Relación_SuperClase	Relaciones Propiedades	

Tabla III.4. Marco Ranuras

Relaciones	CARD. MINIMA	CARD. MAXIMA	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase			Relación_SubClase	Ranura	
SuperClase			Relación_SuperClase	Relación_Estándar Relación_No_Estándar	
Nombre_Relación			Tipo_Conj._Caracteres	Relación	
(*) Ident_Relación	1	1	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*) Origen	1	1	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*) Destino	1	N	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres

Tabla III.5. Marco Relación

Relación_Estándar	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Relación
SuperClase	Relación_SuperClase	Relación_Estándar_Directa Relación_Estándar_Inversa
Nombre_Relación	Tipo_Conj._Caracteres	Relación_Estándar

Tabla III.6. Marco Relación_Estándar

Relación_ Estándar_Directa	CARD. MIN.	CARD. MAX.	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase			Relación_SubClase	Relación_Estándar	
SuperClase			Relación_SuperClase	Relación_SubClase Relación_Instancia	
Nombre_Relación			Tipo_Conj._Caracteres	Relación_Estándar_Directa	
(*) Destino	1	1	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres

Tabla III.7. Marco Relación_Estándar_Directa

Relación_SubClase	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Relación_Estándar_Directa
Nombre_Relación	Tipo_Conj._Caracteres	Relación_SubClase
Relación_Inversa	Tipo_Conj._Caracteres	Relación_SuperClase
Si Añado	Método	Crear_Relación_SubClase (Origen, Destino)
Si Borro	Método	Borrar_Relación_SubClase (Origen, Destino)

Tabla III.8. Marco Relación_SubClase

Relación_Instancia	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Relación_Estándar_Directa
Nombre_Relación	Tipo_Conj._Caracteres	Relación_Instancia
Relación_Inversa	Tipo_Conj._Caracteres	Relación_Elementos_Clase
Si Añado	Método	Crear_Relación_Instancia (Origen, Destino)
Si Borro	Método	Borrar_Relación_Instancia (Origen, Destino)

Tabla III.9. Marco Relación_Instancia

Relación_ Estándar_Inversa	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Relación_Estándar
SuperClase	Relación_SuperClase	Relación_SuperClase Relación_Elementos_Clase
Nombre_Relacion	Tipo_Conj._Caracteres	Relación_Estándar_Inversa

Tabla III.10. Marco Relación_Estándar_Inversa

Relación_SuperClase	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Relación_Estándar_Inversa
Nombre_Relación	Tipo_Conj._Caracteres	Relación_SuperClase
Relación_Inversa	Tipo_Conj._Caracteres	Relación_SubClase
Si Añado	Método	Crear_Relación_SuperClase (Origen, Destino)
Si Borro	Método	Borrar_Relación_SuperClase (Origen, Destino)

Tabla III.11. Marco Relación_SuperClase

Relación_Elementos_Clase	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Relación_Estándar_Inversa
Nombre_Relación	Tipo_Conj._Caracteres	Relación_Elementos_Clase
Relación_Inversa	Tipo_Conj._Caracteres	Relación_Instancia
Si Añado	Método	Crear_Relación_Elementos_Clase (Origen, Destino)
Si Borro	Método	Borrar_Relación_Elementos_Clase (Origen, Destino)

Tabla III.12. Marco Relación_Elementos_Clase

Relación_ No_Estándar	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Relación
SuperClase	Relación_SuperClase	Relación_Fraternal Relación_Disjunto Relación_No_Disjunto Relación_Pertenece Relación_Elementos_Conjunto Relación_Ad_Hoc
Nombre_Relación	Tipo_Conj._Caracteres	Relación_No_Estándar

Tabla III.13. Marco Relación_No_Estándar

Relación_Pertenece	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Relación_No_Estándar
Nombre_Relación	Tipo_Conj._Caracteres	Relación_Pertenece
Relación_Inversa	Tipo_Conj._Caracteres	Relación_Elementos_Conjunto
SI Añado	Método	Crear_Relación_Pertenece (Origen, Destino)
SI Borro	Método	Borrar_Relación_Pertenece (Origen, Destino)
Propiedades	Tipo_Conj._Caracteres	Tipo_Conj._Caracteres

Tabla III.17. Marco Relación_Pertenece

Relación_Elementos_Conjunto	TIPO RANURA	PROPIEDAD GENERAL
SubClase	Relación_SubClase	Relación_No_Estándar
Nombre_Relación	Tipo_Conj._Caracteres	Relación_Elementos_Conjunto
Relación_Inversa	Tipo_Conj._Caracteres	Relación_Pertenece
SI Añado	Método	Crear_Relación_Elementos_Conjunto (Origen, Destino)
SI Borro	Método	Borrar_Relación_Elementos_Conjunto (Origen, Destino)

Tabla III.18. Marco Relación_Elementos_Conjunto

Relación_Ad_Hoc	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Relación_No_Estándar	
SI Añado_Clase	Método	Crear_Clase_Relación_Ad_Hoc (Nombre_Rel., MC_Origen, MC_Destino)	
SI Borro_Clase	Método	Borrar_Clase_Relación_Ad_Hoc (Nombre_Rel., MC_Origen, MC_Destino)	
SI Añado_Instancia	Método	Crear_Relación_Ad_Hoc_Instancia (Nombre_Rel., MI_Origen, MI_Destino, MC_Origen, MC_Destino)	
SI Borro_Instancia	Método	Borrar_Relación_Ad_Hoc_Instancia (Nombre_Rel., MI_Origen, MI_Destino, MC_Origen, MC_Destino)	
(*) Nombre_Relación	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*) Relación_Inversa	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*)Propiedades_Exportables	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*) Simétrica	Tipo_Lógico		[Cierto, Falso]
(*) Transitiva	Tipo_Lógico		[Cierto, Falso]

Tabla III.19. Marco Relación_Ad_Hoc

Propiedades	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Ranuras	
SuperClase	Relación_SuperClase	Propiedades_de_Clase Propiedades_de_Instancia	
(*) *Nombre_Propiedad*	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*) *Definida_En*	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*) *Identificador*	Tipo_Conj._Caracteres		Tipo_Conj._Caracteres
(*) *Cardinalidad_Mínima*	Tipo_Entero_Corto		Tipo_Entero_Corto
(*) *Cardinalidad_Máxima*	Tipo_Entero_Corto		Tipo_Entero_Corto
(*) *Multivaluado*	Tipo_Lógico		If *Cardinalidad_Máxima*>1 Then *Multivaluado* = True Else *Multivaluado* = False
(*) *Representatividad*	Tipo_Real		[0, 1]
(*) *Grado de Certeza*	Tipo_Real		[-1, 1]

Tabla III.20. Marco Propiedades

Propiedades_de_Instancia	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Propiedades	
Tipo_Propiedad	Tipo_Conj._Caracteres	de_Instancia	
Si Añado	Método	Crear_Propiedades_de_Instancia (Marco, Propiedad))	
Si Borro	Método	Borrar_Propiedades_de_Instancia (Marco, Propiedad))	
(*) *Valores_Permitidos*	Tipo_Simple Tipo_Compuesto Marco Programa		Tipo_Simple Tipo_Compuesto Puntero
(*) *Valores_Omisión*	Tipo_Simple Tipo_Compuesto Marco Programa		Tipo_Simple Tipo_Compuesto Puntero
(*) *Precondiciones*	Marco Operación_Lógica Marco Sentencias_Alternativas Marco Programa		Puntero Tipo_Lógico
(*) *Si Necesito*	Marco Sentencia Marco Programa		Puntero
(*) *Si Modifico*	Marco Sentencia Marco Programa		Puntero
(*) *Si Añado*	Marco Sentencia Marco Programa		Puntero
(*) *Si Borro*	Marco Sentencia Marco Programa		Puntero

Tabla III.21. Marco Propiedades_de_Instancia

Propiedades_de_Clase	TIPO RANURA	PROPIEDAD GENERAL	VALORES PERMITIDOS
SubClase	Relación_SubClase	Propiedades	
Tipo_Propiedad	Tipo_Conj._Caracteres	de_Clase	
Si Añado	Método	Crear_Propiedades_de_Clase (Marco, Propiedad)	
Si Borro	Método	Borrar_Propiedades_de_Clase (Marco, Propiedad)	
(*) Propiedad_General	Tipos_Compuestos Tipos_Simples		If *Multivaluado* = True Then Tipos_Compuestos Else Tipos_Simples

Tabla III.22. Marco Propiedades_de_Clase

III.2. METODOS DE LOS MARCOS

En esta sección, se muestran los métodos asociados a las ranuras *Si añado* y *Si borro* de los marcos que representan los conceptos de *Marco Clase* y *Marco Instanciado*.

III.2.1. MARCOS CLASE

Crear_Marco_Clase (Marco)

Precondición:

No Es_Instancia_Marco_Clase (Marco)

Acción:

Crear_Instancia_Marco_Clase (Marco)

Postcondición:

Es_Instancia_Marco_Clase (Marco)

Borrar_Marco_Clase (Marco)

Precondición:

Es_Instancia_Marco_Clase (Marco)

Acción:

Borrar_Instancia_Marco_Clase (Marco)

Postcondición:

No Es_Instancia_Marco_Clase (Marco)

III.2.2. MARCOS INSTANCIADOS

Crear_Marco_Instanciado (Marco)

Precondición:

No Es_Instancia_Marco_Instanciado (Marco)

Acción:

Crear_Instancia_Marco_Instanciado (Marco)

Postcondición:

Es_Instancia_Marco_Instanciado (Marco)

Borrar_Marco_Instanciado (Marco)

Precondición:

Es_Instancia_Marco_Instanciado (Marco)

Acción:

Borrar_Instancia_Marco_Instanciado (Marco)

Postcondición:

No Es_Instancia_Marco_Instanciado (Marco)

III.3. METODOS DE LAS RELACIONES

En esta sección se describen los métodos asociados al conjunto de marcos que representan el concepto *Relación* en la Jerarquía de Conceptos de Marco.

III.3.1. M_RELACION_SUBCLASE

Crear_Relación_SubClase (Origen, Destino)

Precondición:

Es_Instancia_Marco_Clase (Origen) ∧
Es_Instancia_Marco_Clase (Destino) ∧
No Es_Instancia_Relación_SubClase (Origen, Destino) ∧
No Circularidad_Marco_Clase (Destino, Origen) ∧

Acción:

Crear_Instancia_Relación_SubClase (Origen, Destino)
Crear_Relación_SuperClase (Destino, Origen)
Crear_Relaciones_Fraternal (Origen, Destino.hijos)

Postcondición:

Es_Instancia_Relación_SubClase (Origen, Destino)

Borrar_Relación_SubClase (Origen, Destino)

Precondición:

Es_Instancia_Relación_SubClase (Origen, Destino)

Acción:

Borrar_Relaciones_Fraternales (Origen, Destino.hijos)
Borrar_Instancia_Relación_SubClase (Origen, Destino)
Borrar_Relación_SuperClase (Destino, Origen)

Postcondición:

No Es_Instancia_Relación_SubClase (Origen, Destino)

III.3.2. M_RELACION_SUPERCLASE

Crear_Relación_SuperClase (Origen, Destino)

Precondición:

Es_Instancia_Marco_Clase (Origen) ∧
Es_Instancia_Marco_Clase (Destino) ∧
No Es_Instancia_Relación_SuperClase (Origen, Destino) ∧
No Circularidad_Marco_Clase (Origen, Destino)

Acción:

Crear_Instancia_Relación_SuperClase (Origen, Destino)
Crear_Relación_SubClase (Destino, Origen)

Postcondición:

Es_Instancia_Relación_SuperClase (Origen, Destino)

Borrar_Relación_SuperClase (Origen, Destino)

Precondición:

Es_Instancia_Relación_Superclase (Origen, Destino)

Acción:

Borrar_Instancia_Relación_SuperClase (Origen, Destino)
Borrar_Relación_SubClase (Destino, Origen)

Postcondición:

No Es_Instancia_Relación_SuperClase (Origen, Destino)

III.3.3. M_RELACION_INSTANCIA

Crear_Relación_Instancia (Origen, Destino)

Precondición:

Es_Instancia_Marco_Instanciado (Origen) ∧
Es_Instancia_Marco_Clase (Destino) ∧
No Es_Instancia_Relación_Instancia (Origen, Destino) ∧
No Ascendentes_Disjuntos (Origen)

Acción:

Crear_Instancia_Relación_Instancia (Origen, Destino)
Crear_Relación_Elementos_Clase (Destino, Origen)

Postcondición:

Es_Instancia_Relación_Instancia (Origen, Destino)

Borrar_Relación_Instancia (Origen, Destino)

Precondición:

Es_Instancia_Relación_Instancia (Origen, Destino)

Acción:

Borrar_Instancia_Relación_Instancia (Origen, Destino)
Borrar_Relación_Elementos_Clase (Destino, Origen)

Postcondición:

No Es_Instancia_Relación_Instancia (Origen, Destino)

III.3.4. M_RELACION_ELEMENTOS_CLASE

Crear_Relación_Elementos_Clase (Origen, Destino)

Precondición:

Es_Instancia_Marco_Clase (Origen) ∧
Es_Instancia_Marco_Instanciado (Destino) ∧
No Es_Instancia_Relación_Elementos_Clase (Origen, Destino)

Acción:

Crear_Instancia_Relación_Elementos_Clase (Origen, Destino)
Crear_Relación_Instancia (Destino, Origen)

Postcondición:

Es_Instancia_Relación_Elementos_Clase (Origen, Destino)

Borrar_Relación_Elementos_Clase (Origen, Destino)

Precondición:

Es_Instancia_Relación_Elementos_Clase (Origen, Destino)

Acción:

Borrar_Instancia_Relación_Elementos_Clase (Origen, Destino)
Borrar_Relación_Instancia (Destino, Origen)

Postcondición:

No Es_Instancia_Relación_Elementos_Clase (Origen, Destino)

III.3.5. M_RELACION_FRATERNAL

Crear_Relación_Fraternal (Origen, Destino)

Precondición:

Es_Instancia_Marco_Clase (Origen) ∧
Es_Instancia_Marco_Clase (Destino) ∧
Diferente (Origen, Destino) ∧
No Es_Instancia_Relación_Fraternal (Origen, Destino) ∧
Mismo_Padre (Origen, Destino)

Acción:

Crear_Instancia_Relación_Fraternal (Origen, Destino)
Crear_Relación_Fraternal (Destino, Origen)

Postcondición:

Es_Instancia_Relación_Fraternal (Origen, Destino)

Borrar_Relación_Fraternal (Origen, Destino)

Precondición:

Es_Instancia_Relación_Fraternal (Origen, Destino)

Acción:

Borrar_Instancia_Relación_Fraternal (Origen, Destino)
Borrar_Relación_Fraternal (Destino, Origen)

Postcondición:

No Es_Instancia_Relación_Fraternal (Origen, Destino)

III.3.6. M_RELACION_DISJUNTO

Crear_Relación_Disjunto (Origen, Destino)

Precondición:

Es_Instancia_Marco_Clase (Origen) ∧
Es_Instancia_Marco_Clase (Destino) ∧
Diferente (Origen, Destino) ∧
No Es_Instancia_Relación_Disjunto (Origen, Destino) ∧
No Es_Instancia_Relación_No_Disjunto (Origen, Destino)

Acción:

Crear_Instancia_Relación_Disjunto (Origen, Destino)
Crear_Relación_Disjunto (Destino, Origen)

Postcondición:

Es_Instancia_Relación_Disjunto (Origen, Destino)

Borrar_Relación_Disjunto (Origen, Destino)

Precondición:

Es_Instancia_Relación_Disjunto (Origen, Destino)

Acción:

Borrar_Instancia_Relación_Disjunto (Origen, Destino)
Borrar_Relación_Disjunto (Destino, Origen)

Postcondición:

No Es_Instancia_Relación_Disjunto (Origen, Destino)

III.3.7. M_RELACION_NO_DISJUNTO

Crear_Relación_No_Disjunto (Origen, Destino)

Precondición:

Es_Instancia_Marco_Clase (Origen) ∧
Es_Instancia_Marco_Clase (Destino) ∧
Diferente (Origen, Destino) ∧
No Es_Instancia_Relación_No_Disjunto (Origen, Destino) ∧
No Es_Instancia_Relación_Disjunto (Origen, Destino)

Acción:

Crear_Instancia_Relación_No_Disjunto (Origen, Destino)
Crear_Relación_No_Disjunto(Destino, Origen)

Postcondición:

Es_Instancia_Relación_No_Disjunto (Origen, Destino)

Borrar_Relación_No_Disjunto (Origen, Destino)

Precondición:

Es_Instancia_Relación_No_Disjunto (Origen, Destino)

Acción:

Borrar_Instancia_Relación_No_Disjunto (Origen, Destino)
Borrar_Relación_No_Disjunto (Destino, Origen)

Postcondición:

No Es_Instancia_Relación_No_Disjunto (Origen, Destino)

III.3.8. M_RELACION_PERTENECE

Crear_Relación_Pertenece (Origen, Destino)

Precondición:

Es_Instancia_Marco_Clase (Origen) ∧
Es_Instancia_Marco_Clase (Destino) ∧
No Es_Instancia_Relación_Pertenece (Origen, Destino)

Acción:

Crear_Instancia_Relación_Pertenece (Origen, Destino)
Crear_Relación_Elementos_Conjunto (Destino, Origen)

Postcondición:

Es_Instancia_Relación_Pertenece (Origen, Destino)

Borrar_Relación_Pertenece (Origen, Destino)

Precondición:

Es_Instancia_Relación_Pertenece (Origen, Destino)

Acción:

Borrar_Instancia_Relación_Pertenece (Origen, Destino)
Borrar_Relación_Elementos_Conjunto (Destino, Origen)

Postcondición:

No Es_Instancia_Relación_Pertenece (Origen, Destino)

III.3.9. M_RELACION_ELEMENTOS_CONJUNTO

Crear_Relación_Elementos_Conjunto (Origen, Destino)

Precondición:

Es_Instancia_Marco_Clase (Origen) ∧
Es_Instancia_Marco_Clase (Destino) ∧
No Es_Instancia_Relación_Elementos_Conjunto (Origen, Destino)

Acción:

Crear_Instancia_Relación_Elementos_Conjunto (Origen, Destino)
Crear_Relación_Pertenece (Destino, Origen)

Postcondición:

Es_Instancia_Relación_Elementos_Conjunto (Origen, Destino)

Borrar_Relación_Elementos_Conjunto (Origen, Destino)

Precondición:

Es_Instancia_Relación_Elementos_Conjunto (Origen, Destino)

Acción:

Borrar_Instancia_Relación_Elementos_Conjunto (Origen, Destino)
Borrar_Relación_Pertenece (Destino, Origen)

Postcondición:

No Es_Instancia_Relación_Elementos_Conjunto (Origen, Destino)

III.3.10. M_RELACION_AD_HOC

Crear_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Origen, MC_Destino)

Precondición:

No Es_Marco_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Origen, MC_Destino) ∧
Es_Instancia_Marco_Clase(MC_Origen) ∧
Es_Instancia_Marco_Clase (MC_Destino)

Acción:

Crear_Instancia_Marco_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Origen, MC_Destino)
Crear_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Destino, MC_Origen)

Postcondición:

Es_Marco_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Origen, MC_Destino)

Borrar_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Origen, MC_Destino)

Precondición:

Es_Marco_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Origen, MC_Destino)

Acción:

Borrar_Instancia_Marco_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Origen, MC_Destino)
Borrar_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Destino, MC_Origen)

Postcondición:

No Es_Marco_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Origen, MC_Destino)

Crear_Relación_Ad_Hoc_Instancia
(Nombre_Rel.,
MI_Origen, MI_Destino,
MC_Origen, MC_Destino)

Precondición:

Es_Marco_Clase_Relación_Ad_Hoc
(Nombre_Rel., MC_Origen, MC_Destino) ∧
Es_Instancia_Marco_Instanciado (MI_Origen) ∧
Es_Instancia_Marco_Instanciado (MI_Destino) ∧
Es_Instancia (MI_Origen, MC_Origen) ∧
Es_Instancia (MI_Destino, MC_Destino) ∧
No Es_Instancia_Relación_Ad_Hoc
(Nombre_Rel., MI_Origen, MI_Destino, MC_Origen, MC_Destino)

Acción:

Crear_Instancia_Relación_Ad_Hoc
(Nombre_Rel., MI_Origen, MI_Destino, MC_Origen, MC_Destino)
Crear_Relación_Ad_Hoc_Instancia
(Nombre_Rel.,
MI_Destino, MI_Origen,
MC_Destino, MC_Origen)

Postcondición:

Es_Instancia_Relación_Ad_Hoc
(Nombre_Rel., MI_Origen, MI_Destino, MC_Origen, MC_Destino)

Borrar_Relación_Ad_Hoc_Instancia
(Nombre_Rel.,
MI_Origen, MI_Destino,
MC_Origen, MC_Destino)

Precondición:

Es_Instancia_Relación_Ad_Hoc
(Nombre_Rel., MI_Origen, MI_Destino, MC_Origen, MC_Destino)

Acción:

Borrar_Instancia_Relación_Ad_Hoc
(Nombre_Rel., MI_Origen, MI_Destino, MC_Origen, MC_Destino)

Borrar_Relación_Ad_Hoc_Instancia
(Nombre_Rel.,
MI_Destino, MI_Origen,
MC_Destino, MC_Origen)

Postcondición:

No Es_Instancia_Relación_Ad_Hoc
(Nombre_Rel., MI_Origen, MI_Destino, MC_Origen, MC_Destino)

III.4. METODOS DE LAS PROPIEDADES

Se pasa a especificar los procedimientos asociados a las ranuras que representan métodos en los marcos que representan los conceptos de *Propiedades de_Clase* y *Propiedades de_Instancias*.

III.4.1. PROPIEDADES DE CLASE

Crear_Propiedades_de_Clase (Propiedad, Marco)

Precondición:

No Es_Instancia_Propiedades_de_Clase (Propiedad, Marco)

Acción:

Crear_Instancia_Propiedades_de_Clase (Propiedad, Marco)

Postcondición:

Es_Instancia_Propiedad_de_Clase (Propiedad, Marco)

Borrar_Propiedades_de_Clase (Propiedad, Marco)

Precondición:

Es_Instancia_Propiedades_de_Clase (Propiedad, Marco)

Acción:

Borrar_Instancia_Propiedades_de_Clase (Propiedad, Marco)

Postcondición:

No Es_Instancia_Propiedades_de_Clase (Propiedad, Marco)

III.4.2. PROPIEDADES DE INSTANCIA

Crear_Propiedades_de_Instancia (Propiedad, Marco)

Precondición:

No Es_Instancia_Propiedades_de_Instancia (Propiedad, Marco)

Acción:

Crear_Instancia_Propiedades_de_Instancia (Propiedad, Marco)

Postcondición:

Es_Instancia_Propiedad_de_Instancia (Propiedad, Marco)

Borrar_Propiedades_de_Instancia (Propiedad, Marco)

Precondición:

Es_Instancia_Propiedades_de_Instancia (Propiedad, Marco)

Acción:

Borrar_Instancia_Propiedades_de_Instancia (Propiedad, Marco)

Postcondición:

No Es_Instancia_Propiedades_de_Instancia (Propiedad, Marco)